INDUS CORPORATION
Knowledge-Based Solutions

3i Systems
Innovation Internet Intelligence

Department of Commerce Intranet Architecture

Prepared for:

Dale Lanser, COR

U.S. Department of Commerce

OCIO, Office of Information Systems

Prepared by:

INDUS Corporation

1953 Gallows Road

Vienna, Virginia 22182

DOC Contract Number:	50CMAA900048
Task Order Number:	ISE00032

Task 2: Standards and Protocols Research

November 13, 2000

Table of Contents

United States Department of Commerce Intranet Architecture Overview

1 Introduction

This document explains the high level architecture for the Department of Commerce intranet. It explains the primary components of the system, how they are organized onto physical machines, and how they communicate with each other. It details what support software each component needs in order to operate correctly.

This document does not provide hardware specifications for the various server machines mentioned. It does not attempt to specify the design details of the various sub-components of each system component. Internal design issues are occasionally mentioned to aid in the understanding of the overlying architecture.

2 Overview of the Architecture

The DOC Intranet can be divided into large architectural components and the communication channels among them. Figure 1 depicts the main components of the intranet:

1) WAP server for authentication,

2) Directory server to store user and application information, and

3) One or more application servers.

A primary application on the intranet is the intranet portal. It was chosen to represent a typical application in this diagram. Figure 1 shows the three components arranged on top of three server class machines. Within each machine are the various software sub-systems that are required for the component to function correctly within the intranet. Each larger component could be one or more machines working together to provide the given functionality. The dotted lines in Figure 1 show possible machine divisions.

Not depicted in Figure 1 is the user who accesses the intranet via a web browser. The browser must supports cookies.

2.1 Communication Channels

The solid arrow-tipped lines indicate communication pathways and protocols among the various pieces of the system. Both intra-machine and inter-machine channels are depicted.

There are three primary communication protocols used within the intranet:

1) WAP – Web user Authentication Protocol

2) IACP – Inter Application Communication Protocol

3) LDAP – Lightweight Directory Access Protocol

4) AJP – Advanced Jserv Protocol

WAP and IACP are proprietary and are discussed in detail in ***Appendix A: Web-user Authentication Protocol*** and ***Appendix F: DOC Intranet Inter-Application*** Communication Protocol respectively. Both sit on top of the HTTP protocol. LDAP is an industry standard for communicating with a hierarchical text-based database. AJP is a protocol developed to facilitate communication between a web server adapter (in this case mod_jserv) and a servlet container (in this case Tomcat).

In the sections below, each component is addressed in more detail including what support software is required and how the component might be laid out across several different physical computers.

3 The WAP Server

3.1 Overview

The WAP server is labeled on Figure 1 as ***login.doc.gov***. The WAP server works in conjunction with the Directory Service to provide the authentication mechanism for the intranet. It provides for the single sign-on capability for the intranet.

Typically, users do not contact the WAP server directly; rather, they access a departmental application. If the user has not been authenticated, the application sends a redirect to the user's browser pointing it to the WAP server. The WAP server then provides the user with a login page.

The user enters their username and password across an SSL connection to the WAP server. The WAP server looks up the username in the Directory service component across an LDAP connection. The Directory service returns the user's IID number. This number is then used to verify the user's password in a local disk file. Upon successful login, the WAP server redirects the user back to the application with an authentication cookie. See the Web-user Authentication Protocol document for a detailed discussion on how this authentication occurs.

3.2 The WAP Password File

The WAP server maintains a local disk file that contains three fields:

1) User's IID number,

2) User's encrypted password, and

3) Enabled bit.

The user IID number is a unique number assigned to every user on the intranet. An IID is created by an administrator when a user is added to the system. It should never change. The passwords are encrypted using the NBS DES algorithm (used by Unix) or another secure encryption algorithm. The last field in the file is a boolean value representing whether or not the account has active status. Disabled accounts cannot be logged in to.

3.3 Security

Because the WAP server is the central security system for the Intranet, the machine it runs on must to be as secure as possible. A disk file resides on the server that contains all of the intranet users' IID numbers and their encrypted passwords. Anyone with access to this file can potentially infiltrate the intranet.

Only SSL connections can be made to the WAP server to prevent the sniffing of user passwords when they are sent from the user's browser across the intranet (or internet) to the WAP server. After a user's initial login, their password never needs to be sent across the network again.

3.4 Support Software

The WAP authenticator is a Java servlet. It needs a servlet container in order to run. Additionally, the authenticator is accessed via the WAP protocol that sits on top of the HTTP protocol. Therefore, the WAP server machine needs to have a web server running on it to provide the entry point for requests for authentication from the various intranet applications.

Figure 1 shows the WAP application as a servlet inside a servlet container communicating with a web server with a servlet adapter. AJP is used over this communication channel.

3.5 The Hardware

The WAP server machine is shown in Figure 1 as a single machine. It is possible to distribute it across two or more separate physical machines. The dotted line in the figure indicates the division point. The servlet container can exist and execute on one physical machine and the web server and jserv adapter may exist on another. In a heavily loaded environment, the web server depicted may actually be several web servers (each on a separate machine) all configured to operate together. The same load-balancing mechanism can be arranged for the servlet containers as well.

4 The Directory Server

The directory server contains the directory service application for the intranet. It is labeled in figure 1 as ***directory.doc.gov***. This service stores information about the intranet's users including their IID numbers, contact information, and preferences for intranet applications. It also stores alternate usernames (aliases) for the user so legacy applications can be accessed without logging into them separately. The directory also stores information about the various applications on the intranet such as which applications are trusted and which connections have to be SSL.

As mentioned, the directory service is an LDAP directory.

The directory service is accessed by the WAP server and by applications directly. In order to access (and maintain) the information in the directory, two applications (servlets) exist that interface with the service. The first, the Directory Management application, allows users to access information about themselves and other users. Users can also use this program to update information about themselves. The second application is the Group Services application. It is a helper application used by other intranet applications to perform more complex lookups into the Directory service hierarchical structure.

4.1 The Directory Management Application

The Directory Management application is a Java servlet that allows users to access and update information in the intranet directory. It is envisioned to have a database backend that is used to enforce user permissions and roles in the directory. Users have certain roles with certain access permissions. Users can modify most of the information about themselves and users designated as administrators can modify information about other users as well.

This application performs simple queries in the intranet directory. For more complex queries and lookups, this application makes requests to the Group Services application.

Figure 1 shows that the Directory Management application interfaces to the Directory Service using LDAP and with the Group Services application using IACP.

4.2 The Group Services Application

The Group Services application is also a Java servlet. However, users do not directly access this application. It is a helper application for accessing information in the Directory service that may not be easily obtained through the LDAP interface. This application is accessed using the Inter-Application Communications Protocol, IACP (over HTTP).

4.3 Support Software

Both of these applications are servlets and thus are accessed over HTTP. Therefore, both applications need to be paired with a web server as well as a servlet container for correct operation. Figure 1 shows a web server and a servlet container on the directory server machine.

4.4 Hardware

The directory component may be distributed among more than one machine, separating the servlets and the web servers from each other. Also, the LDAP Directory service could reside on a separate machine. Again, the dotted lines in Figure 1 indicate these division points. This component may exist on four separate machines.

4.5 Summary

The Directory Management application and the Group Service application can be thought of as regular intranet applications. The Group Services application is unique in that most applications on the intranet will be communicating with it to obtain information about users.

5 The Portal Application

The last component depicted in Figure 1 is the portal server (***portal.doc.gov***). The portal application represents a typical application in the intranet. It performs almost every task that any other application might perform, so it is a good sample application. It is envisioned to be a modified version of Jetspeed, the Java-Apache open source portal product. As such, it needs a servlet container and a web server in order to integrate into the Intranet environment.

The portal server communicates with various applications on the intranet and creates a portal page of information for a logged in user. As such, the portal application interacts with the Directory Management application via the IACP protocol. It also communicates directly with the directory service to get configuration information about the various applications that are included in the portal.

6 Other Applications

Other applications behave as the ones already outlined. Published APIs make the design and implementation of intranet applications easier. The API definitions are in ***Appendix B: WAP Client API*** for the DOC Intranet and ***Apendix G: DOC Inter-Application Communication*** Protocol API. APIs exist for the three most common web development languages (Java, Perl, and PHP).

7 Architecture Features

7.1 Single Sign-On

The single sign-on feature of the Intranet is implemented using the WAP server and cookies.

When an application, such as the portal server, is accessed by a user, the application checks to see if the user has logged in. If the user has not yet logged into the intranet, the application will redirect the user's browser to the WAP server. The user may log in and be given a cookie to send to the application. This cookie stores the user's authentication information and is automatically sent whenever the user accesses an application on the Intranet. This allows users to log into the intranet once and maintain an authentication cookie around to log into other applications after that. See ***Appendix A: Web-user Authentication Protocol*** for a step-by-step description of this process.

The important point to note is that the application does not communicate with the WAP server directly. Rather, the user’s browser makes a request to the WAP server on behalf of the application that the user is attempting to connect to.

7.2 Inter Application Communication

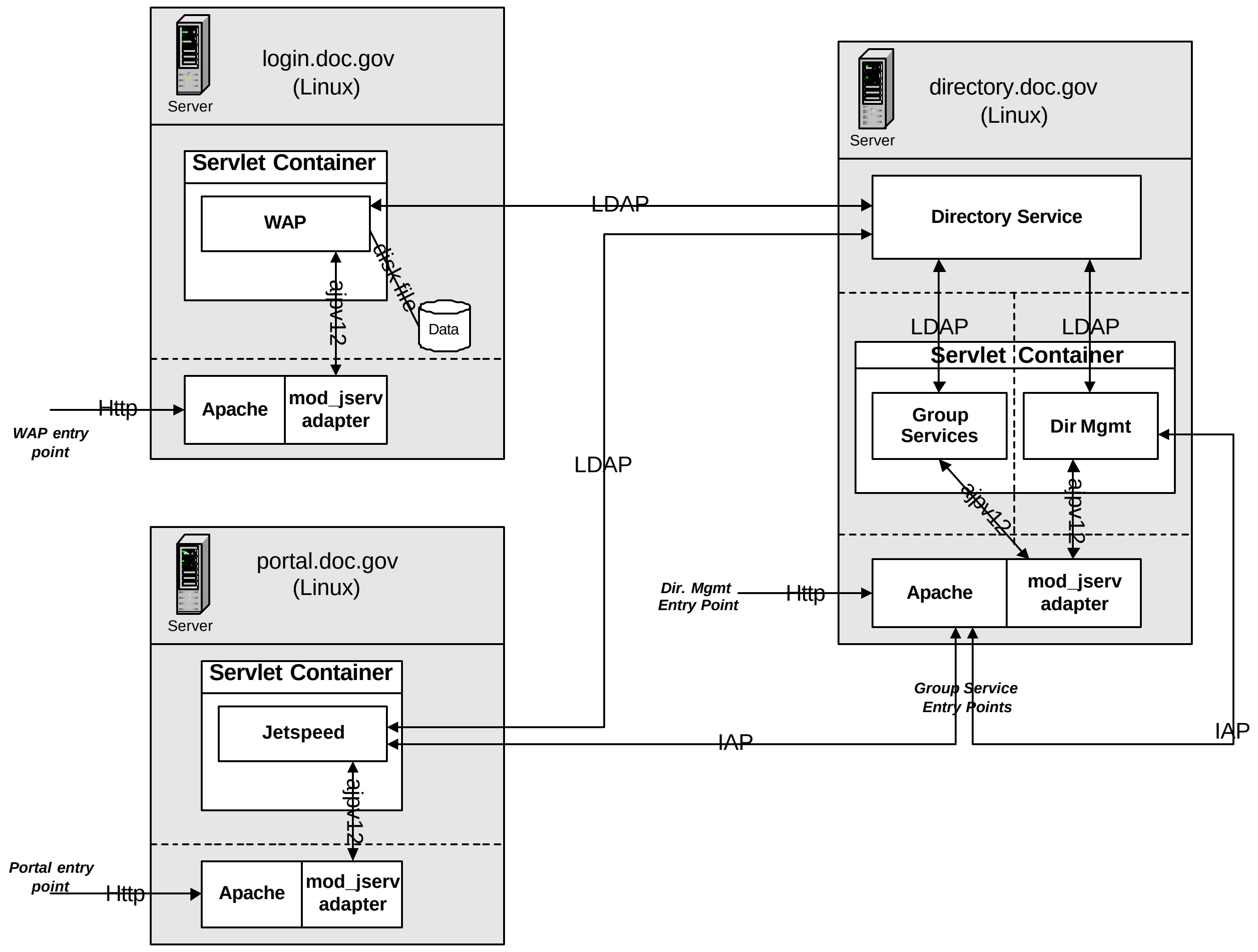
Applications are able to communicate with each other via the Inter-Application Communication Protocol, IACP. This protocol allows applications that trust each other to share information about the intranet users. See the [IACP] document for a detailed description of this protocol and the [IACP-API] document for a description of the APIs for Java, Perl, and PHP for inter-application communication.

8 Other Issues

Hardware requirements must be defined.

Addition of proxy servers should be considered.

Figure 1 Intranet Components



Appendix A: Web-user Authentication Protocol

Version: 1.0
Revised: 2000.11.12

1 Introduction

This document describes the Web-user Authentication Protocol (WAP). WAP is a protocol that can be used to authenticate users to untrusted web-based applications over an insecure network.

2 Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]. An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be "conditionally compliant."

For WAP to function, three components are required: a web browser for the user, an application, and an authenticator. A directory service should be used in conjunction with WAP to provide user identification to the authenticator and applications.

2.1 Protocol Requirements

This protocol is designed to:

- Allow for single-sign-on for all applications using the WAP.
- Allow for users to divulge passwords only to an authenticator, never to an application.
- Disallow applications from using any user’s authentication information for authenticating against another application.
- Allow applications, at their own discretion, to provide a “public face” level of service to users who have declined to authenticate.

2.2 Web Browser Requirements

The web browser MUST support cookies, as described in [RFC2109]. Cookies allow users' authentication information to persist over multiple applications (one login per session).

The web browser MUST support HTTP Redirection.

The web browser MUST support SSL, as described in [SSL].

2.3 Application Requirements

The application MUST support HyperText Transfer Protocol (HTTP), as described in [RFC2616]. HTTP is used to service users' requests.

The application MUST support setting and receiving cookies, as described in [RFC2109]. Cookies are one method for passing user authentication information from the authenticator to the application.

The application SHOULD support SSL, as described in [SSL].

The application SHOULD support public key encryption as defined in [CRYPTO].

An application's level of support for the above requirements will determine the level of security an application can expect by implementing WAP and using a WAP authenticator.

2.4 Authenticator Requirements

The authenticator MUST support setting and receiving cookies, as described in [RFC2109]. Cookies are used to persist user authentication information across multiple requests from the user to the authenticator and applications.

The authenticator MUST support HyperText Transfer Protocol (HTTP), as described in [RFC2616]. HTTP is used to service users' requests.

The authenticator MUST support SSL as described in [SSL].

The authenticator MUST support public key encryption as defined in [CRYPTO].

3 Terminology

anonymous application ticket

> An application ticket that does not include a user IID. These are provided to users who decline to supply a username and password, and instead wish to access applications 'public face'.

application

> A system that provides a service to users through an HTML interface via HTTP, as defined in section 6.0.

application IID

> The intranet ID (IID) associated with a specific running instance of an application.

application ticket

> A data element that holds the authentication information for a user. The authentication information is time-stamped, and contains rights for a user to access a particular application. Application tickets are stored on a user's browser as cookies.

authenticator

> A specialized application that provides user authentication services to applications, as defined in section 5.0.

browser, or web browser

> A standard web browser, as defined in section 2.1.

CGI variable

> A POST or GET parameter used in web requests. See [RFC2616].

cookie

> Data that is passed from an application to a web browser, stored by the web browser, and returned to an application at a future time. As described in [RFC2109].

directory service

> A service that provides user and application information. The directory service also stores the public keys of all applications.

Intranet ID (IID)

> A unique string which identifies an object maintained in the directory service. For the purpose of this document, IIDs identify either users or application instances.

legacy application

> A application which does not support user identification through the means of the user's intranet ID (user IID).

legacy application username

> A login name used by a user to identify himself to a legacy application.

reference implementation

> This protocol will initially be developed for the Department of Commerce's Intranet. This initial product is referred herein as the reference implementation.

session ticket

A data element that holds WAP authentication information for a user. The authentication information is time-stamped. Session tickets are stored on a user's browser as cookies.

user

A person using a web browser and the authenticator to access one or more applications.

user IID

The Intranet ID (IID) identifying a user.

WAP username (Intranet username)

A username assigned to a user's intranet account. This username can be changed, but is always associated with the user's unique user IID.

WAP password

A password assigned to a user's intranet account. The password may be changed by the user. The password is always associated with a single user IID.

4 Protocol Overview

To authenticate users, WAP is implemented on an authentication server and application servers. The authenticator authenticates users by WAP username and WAP password over an SSL connection, provides application tickets to users, and provides applications with the ability to validate sessions through cryptographic verification of the session ticket digital signatures.

WAP additionally provides a mechanism to time-out user's session tickets through user inactivity, and to renew tickets prior to expiration in a method that is transparent to users.

From the user's perspective, an authenticated session usually beings when the user attempts to access an application with a web browser. When the application does not find an application ticket in the set of cookies and CGI variables presented by the web browser, the application will consider the user to be not authenticated, and will return a redirect to the authentication ticket service login page.

If the authentication server finds a valid session ticket, it verifies the authenticity of the session ticket using its own private key to verify the session ticket digital signature, and verifies that the conditions of use presented in the session ticket are met. The single condition of use listed on the session ticket is an "expire time" beyond which the ticket is no longer valid. The session ticket also contains the Intranet ID (IID) for the user.

With a valid session ticket in hand, the authenticator creates an application ticket allowing the user to talk to the referring application. The application ticket consists of the user's IID, the application's IID, the source IP address of the web browser connection, and the renew time for the ticket. If the application is a legacy application requiring an application-specific username rather than a user IID, the authenticator looks up the appropriate application username for this user and this app, and includes it in the application ticket.

This ticket is forwarded to the application via CGI variables and cookies. Upon receiving the application ticket, the application can verify its authenticity by checking the digital signature using the authenticator's public key.

When presented the login page by the authenticator, the user can elect not to authenticate by selecting the cancel option. In this case, a special application ticket called an anonymous application ticket is created. Anonymous application tickets lack both a user IID field and an application username field. The authenticator does not create a session ticket when an anonymous application ticket is created. When presented with an anonymous application ticket, applications may decide to either reject the anonymous access, or present a "public face" interface to the user.

5 Authenticator

The authenticator is responsible for user login, providing authentication information to applications, and renewing session tickets. This functionality is provided by the ticket service. In addition, the authenticator is responsible for providing a service for adding, deleting and modifying users and passwords. This functionality is provided by the password service.

Each of the authenticator's functions is a service accessible through a URL, using HTTP or HTTPS, HTML forms or CGI variables, and cookies. Both authenticator services are described in detail.

5.1 Ticket Service

5.1.1 Overview

The ticket service authenticates users by WAP username and WAP password. If the user authenticates correctly, the ticket service creates a session ticket and an application ticket, as well as a digital signature for both the session and application tickets. Both the session ticket digital signature and the application ticket digital signature are created with the authenticator's private key. If the user cancels the login, the ticket service creates only an anonymous application ticket (and application ticket digital signature), but no session ticket.

The session ticket is a simple ticket that is used only by the authenticator during later authentication requests by the same user. During the initial successful login, the authenticator returns the session ticket and its digital signature as cookies to the users browser. Later, when the user's browser contacts the authenticator for an application ticket renewal, the authenticator can verify that the user's browser has a session ticket with a valid digital signature, and can check to ensure that the session ticket has not expired. If the session ticket has expired, the authenticator redirects the user's browser to a page where the user can re-login. If the session ticket has not expired, than a new lease is granted to the user's browser by pushing a new session ticket cookie (and digital signature) with a later expiration date to the browser. In the same transaction, a new application ticket (and digital signature) with a later renew date is pushed via CGI variables to the browser for use by the referring application.

The following table defines the content of the session ticket.

Session ticket format	
Name	**Value**
USER_IID	The user-intranet id uniquely identifying this user on the Intranet
EXPIRE	The date-time stamp when this session is no longer valid

Session tickets are only created when a user correctly submits a valid username and password. Canceling the login page causes the authenticator to create an anonymous application ticket, but no session ticket. The session ticket is formatted simply as a list name=value pairs, separated by the pipe character. For example, a user might have a session ticket of the following form

```
USER_IID=36925|EXPIRE=Fri, 31 Dec 1999 23:59:59 GMT
```

The session ticket and digital signature are returned to the browser with setCookie header lines of the following form.

```
Set-Cookie: SESSION_TICKET_TXT=<plaintext session ticket>;
Set-Cookie: SESSION_TICKET_SIG=<session ticket dig. signature>
```

In addition to the session ticket, the authenticator creates an application ticket for the application that redirected the user's browser to the authenticator's login page. This application ticket contains the user's IID, the application's IID, the IP address of the user's browser (or proxy), and the time after which the application ticket needs to be renewed. If the user declines to authenticate, the USER_IID field is not included. A missing USER_IID field defines an anonymous application ticket. If the application is a legacy application requiring a application-specific username rather than a USER_IID, the generated ticket includes both a USER_IID and an APP_USER field.

The following table defines the content of the application ticket.

Application ticket format	
Name	**Value**
USER_IID	The user-intranet id uniquely identifying this user on the Intranet. This field is not present in anonymous application tickets.
APP_USER	For legacy applications, the username with which this user authenticates to the application. This field is only included with tickets created for legacy applications.
APP_IID	The application intranet id uniquely identifying this application on the Intranet
USER_IP	The IP address from which the browser connected to WAP. May be the browser's IP or the proxy server's IP.

RENEW	The date-time stamp after which the session ticket should be renewed.

The application ticket is formatted as name=value pairs, separated by a pipe character. Below is a sample ticket.

```
USER_IID=3692|APP_USER=john_smith|APP_IID=00032|USER_IP=192.168.
1.128|RENEW= Fri, 31 Dec 1999 23:54:59 GMT
```

The application ticket is digitally signed using the authenticator's private key. The plaintext application ticket and its digital signature are sent to the browser as CGI variables of the following form.

```
rmt_cookie=APP_TICKET_TXT:<plaintext application ticket>
rmt_cookie=APP_TICKET_SIG:<application ticket dig. signature>
```

Upon successful authentication, the user's browser is redirected to the application that initially redirected the user to the login page. By default, this is the URL of the referer header line. However, if the application has provided a RET_URL CGI variable, that value is used as the redirection page. Applications SHOULD provide the RET_URL CGI variable to positively notify the authenticator of the redirect target.

Before redirecting the browser to RET_URL or header.referer, the authenticator verifies that the redirection URL is a valid redirection URL for this application. In the referenced implementation, this is done by consulting the directory service.

5.1.2 Request Input Parameters

The URL for accessing the reference implementation ticket service is https://login.doc.gov.

The ticket service accepts requests via HTTPS. The following CGI variables are processed. All other present CGI variables are included unchanged as CGI GET variables on the response.

CGI Variables processed by the Ticket Service

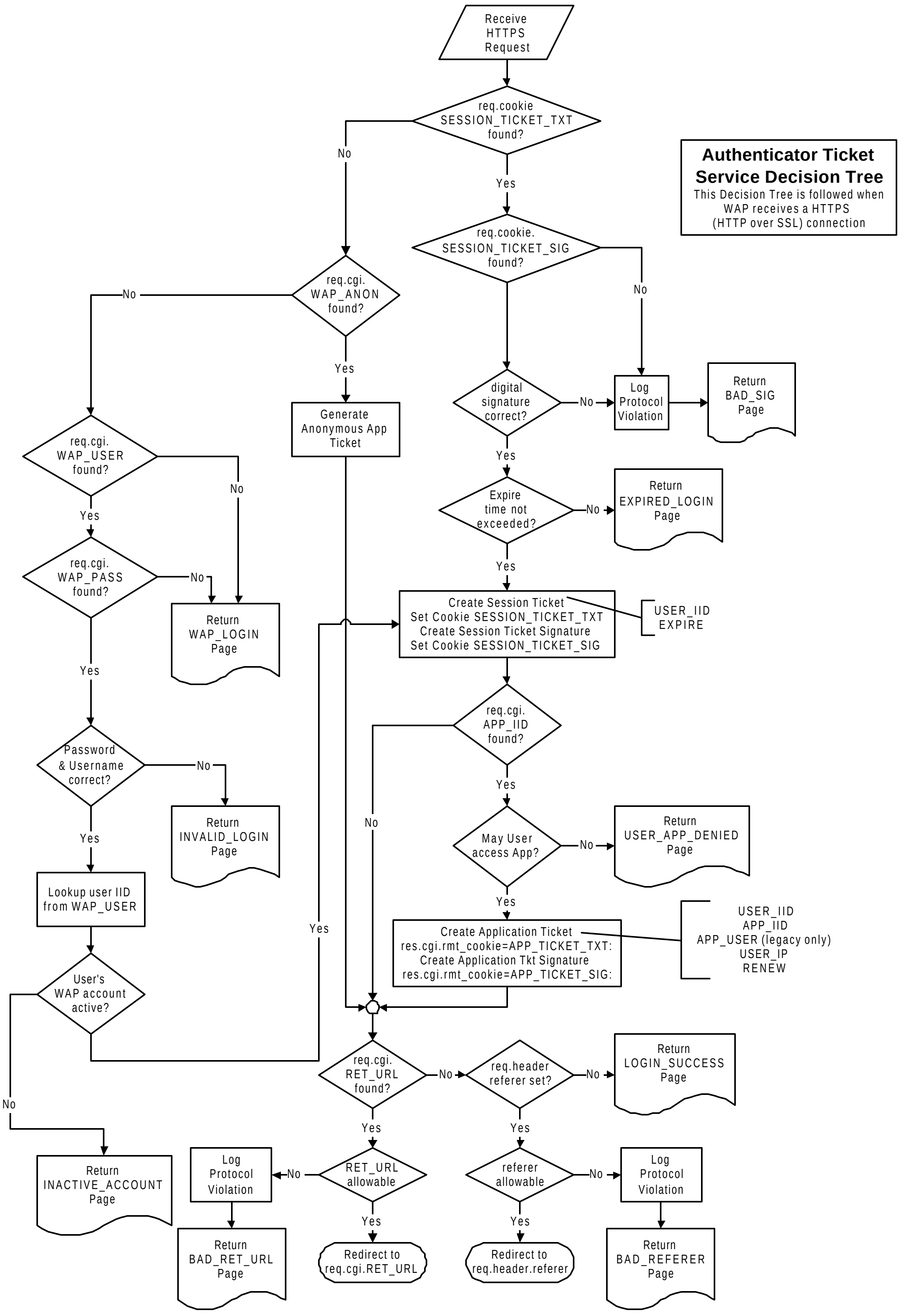
Name	**Value**
WAP_ANON	A flag indicating that the user has selected the cancel option from the login page. If this flag is present, an anonymous ticket will be created for the user.
WAP_USER	The username the user provides to identify himself to the authenticator. There is a one to one relationship between WAP_USER and user IID.
WAP_PASS	The user's WAP password.
APP_IID	The application intranet id uniquely identifying the refering application on the Intranet
RET_URL	The URL that the WAP server should redirect the user's browser to once the user is authenticated.

The ticket service checks for the following cookies to determine if the user is currently logged in.

Cookies accepted by the Ticket Service	
Name	**Value**
SESSION_TICKET_TXT	Encrypted Session Ticket
SESSION_TICKET_SIG	Digital signature of the session ticket

5.1.3 Execution

The execution of the ticket service is described using inputs, processes, decisions, and output. A flowchart representing the flow of execution is provided in the figure on the next page.


Authenticator Ticket Service Decision Tree
This Decision Tree is followed when WAP receives a HTTPS (HTTP over SSL) connection
Receive HTTPS Request
req.cookie SESSION_TICKET_TXT found?
No
Yes
req.cookie. SESSION_TICKET_SIG found?
No
req.cgi. WAP_ANON found?
No
Yes
Generate Anonymous App Ticket
digital signature correct?
No
Log Protocol Violation
Return BAD_SIG Page
Yes
Expire time not exceeded?
No
Return EXPIRED_LOGIN Page
Yes
Create Session Ticket
Set Cookie SESSION_TICKET_TXT
Create Session Ticket Signature
Set Cookie SESSION_TICKET_SIG
USER_IID
EXPIRE
req.cgi. WAP_USER found?
No
Yes
req.cgi. WAP_PASS found?
No
Return WAP_LOGIN Page
Yes
Password & Username correct?
No
Return INVALID_LOGIN Page
Yes
Lookup user IID from WAP_USER
User's WAP account active?
Yes
No
Return INACTIVE_ACCOUNT Page
req.cgi. APP_IID found?
No
Yes
May User access App?
No
Return USER_APP_DENIED Page
Yes
Create Application Ticket
res.cgi.rmt_cookie=APP_TICKET_TXT:
Create Application Tkt Signature
res.cgi.rmt_cookie=APP_TICKET_SIG:
USER_IID
APP_IID
APP_USER (legacy only)
USER_IP
RENEW
req.cgi. RET_URL found?
No
req.header referer set?
No
Return LOGIN_SUCCESS Page
Yes
RET_URL allowable
No
Log Protocol Violation
Return BAD_RET_URL Page
Yes
Redirect to req.cgi.RET_URL
Yes
referer allowable
No
Log Protocol Violation
Return BAD_REFERER Page
Yes
Redirect to req.header.referer

Commentary on Authenticator Ticket service Decision Tree

Receive HTTPS request	This is the starting point for all transactions. A HTTPS (SSL only) connection is made to the well-known WAP Authenticator Ticket service URL.
req.cookie.SESSION_TICKET_TXT found?	Check to see if the user submitted a session ticket.
req.cookie.SESSION_TICKET_SIG found?	Check to see if the user submitted a signature for the session ticket. Session tickets are invalid without a valid signature..
req.cgi.WAP_ANON found?	Check to see if the anonymous flag has been set. This flag is set when the user declines to authenticate by clicking the Cancel button on the WAP_LOGIN page. Anonymous users receive an anonymous application ticket and no session ticket.
Generate Anonymous App Ticket	An anonymous application ticket has no USER_IID field, and no APP_USER field. Users who have not logged into the intranet receive anonymous application tickets.
req.cgi.WAP_USER found?	The authenticator examines the request to see if the CGI variable USER_IID has been included. If it has not been included, then it is not a request coming from the login page. Normal processing ensues.
req.cgi.WAP_PASS found?	The authenticator examines the request to see if the CGI parameter WAP_PASS has been included. If it has not been included, then it is not a request coming from the login page. Normal processing ensues.
Return WAP_LOGIN Page	The user is presented a form that prompts them to supply username and password. These values are sent to the authenticator as CGI variables named WAP_USER and WAP_PASS. Also, a cancel button is present, which when clicked must set the CGI variable WAP_ANON.
Password & Username correct?	The authenticator verifies the username and password match. The password should be stored in a hashed format to reduce damage if compromised. The reference implementation will

	store the hashed passwords, along with user IID and account disable flag, in a local file.
Return INVALID_LOGIN page	This page is returned if the user does not exist or if the password is incorrect. It should either have a link back to the login page or a form to resubmit the username/password information.
Lookup user IID for WAP_USER	The directory service is queried to find a user Intranet ID (IID) matching this username.
User's WAP account active?	Administrators can disable any user's WAP account, which effectively prevents the user from accessing any WAP-enabled account using WAP authentication. This checks the locally stored flag to see if this user's account has been disabled.
Return INACTIVE_ACCOUNT page	A page notifying the user that their Intranet login has been disabled.
Digital Signature Correct?	Using the WAP public key, verify the digital signature contained in SESSION_TICKET_SIG against the plaintext in SESSION_TICKET_TXT.
Log Protocol Violation	An invalid ticket has been received. All such attempts should be logged so that administrators can take appropriate action.
Return BAD_SIG Page	A page should be returned to give the user further guidance on appropriate actions if the SESSION_TICKET_SIG is not genuine. The contents will depend on site preferences – users could be notified that the ticket was rejected, or a generic 'Login Failed' message could be displayed.
Expire Time not exceeded?	The EXPIRE attribute is retrieved from the authenticated SESSION_TICKET. It is compared with the current time on the server. If the current time is earlier than the EXPIRE time, then the session is still valid, and new application and session tickets should be created for the user. If the EXPIRE time has elapsed, then the user should be presented a page noting this, and linking back to the login page.

Return EXPIRED_LOGIN Page	This page notifies the user that their session has timed out due to inactivity, and presents a hyperlink or a login form that allows them to reestablish their session.
Create Session Ticket Set Cookie SESSION_TICKET_TXT Create Session Ticket Signature Set Cookie SESSION_TICKET_SIG	The user has a valid session ticket, and has had recent activity as defined by this protocol. Construct a new session ticket with a new EXPIRE time. Setup the SESSION_TICKET_TXT and SESSION_TICKET_SIG cookie so that they are pushed to the user's browser with the next page.
Req.cgi.APP_IID found?	Check the incoming request to see if a CGI variable named APP_IID is included. If it is, then the user was referred to the authenticator by that app, possibly due to a application ticket needing to be renewed, and possibly because the referring app had no application ticket at all for this user.
May user access App?	Determine if the user has an account managed by WAP for this application. This process is considered to be implementation dependent. The reference implementation will consult the directory service to make this determination.
Return USER_APP_DENIED page	This page notifies the user that his access to the application is not managed by WAP. The new SESSION_TICKET is sent with this page to update the EXPIRE timestamp for the users session.

Create Application Ticket
res.cgi.rmt_cookie=APP_TICKET_TXT:…
Create Application Ticket Signature
res.cgi.rmt_cookie=APP_TICKET_SIG:…
	The user has been authenticated as being able to communicate with this application. Create an application ticket and ticket signature for this app, and setup the APP_TICKET_TXT and APP_TICKET_SIG rmt_cookie CGI variables to be sent with the next page.

Req.gci.RET_URL found?	Check the request to see if it contains a CGI variable named RET_URL. This is the URL that the referring application wishes us to redirect the user to.

RET_URL allowable?	Check to see if the RET_URL URL is a valid URL for this application. The reference implementation will consult the directory service to make this determination.
Redirect to ret.cgi.RET_URL	Redirect the user to the RET_URL, with all CGI variables and headers set previously in this tree.
Return BAD_RET_URL page	This page notifies the user that the referring application resides at an unregistered URL and cannot be redirected to.
Req.header.referer set?	Check the request to see if it contains a referrer header. The referrer header can be used as an alternate to the RET_URL variable. If present, we should redirect the user back to the referring URL.
referrer allowable	Check to see if the referrer URL is a valid URL for this application. The reference implementation will consult the directory service to make this determination.
Redirect to req.header.referer	Redirect the user to the referring URL, with all CGI variables and headers set previously in this tree.
Return BAD_REFERER page	This page notifies the user that the referring application resides at an unregistered URL and cannot be redirected to.
Return LOGIN_SUCCESS Page	A page indicating the login was successful should be presented if we cannot ascertain a location to redirect the user's browser to.

5.1.4 Response Output

Pages Returned by the Authenticator Ticket Service

WAP_LOGIN page

- WAP_USER input field
- WAP_PASS input field
- login button
- cancel button, which sets the WAP_ANON CGI variable

- RET_URL hidden field, if available
- APP_ID hidden field, if available

INVALID_LOGIN page

- Text message informing user that login attempt was invalid
- WAP_USER input field
- WAP_PASS input field
- login button
- cancel button, which sets the WAP_ANON CGI variable
- RET_URL hidden field, if available
- APP_ID hidden field, if available

EXPIRED_LOGIN page

- Text message informing user that the session has expired due to inactivity
- WAP_USER input field
- WAP_PASS input field
- login button
- cancel button, which sets the WAP_ANON CGI variable
- RET_URL hidden field, if available
- APP_ID hidden field, if available

USER_APP_DENIED page

- Site dependent message informing the user that they are not currently configured to access this application.
- Button to allow the user to continue anonymously as an unauthenticated user.

INACTIVE_ACCOUNT

- Message informing the user that their Intranet account has been disabled

LOGIN_SUCCESS page

- Message informing the user that they have successfully logged into the Intranet.

BAD_SIG page

- Message informing user that the SESSION_COOKIE submitted failed the digital signature test.

BAD_RET_URL

- Message informing user that the return application URL is invalid for the application.

BAD_REFERER

- Message informing user that the return application URL is invalid for the application.

Redirects returned by the Authenticator Ticket Service

Redirect to req.cgi.RET_URL

- Redirect to the application the user is attempting to authenticate to

Redirect to req.header.referrer

- If req.cgi.RET_URL is not set, then we can redirect to req.header.referrer, if available.

Authenticator Output

Once the authenticator has generated the application and session keys, it must redirect the user to a URL under the control of the referring application. The authenticator either redirects to the URL specified in the request's RET_URL CGI variable, or the URL specified in the request's referrer header. If the authenticator finds either of these URL's, the authenticator is responsible for first verifying that the URL is an allowable URL for the application. The reference implementation consults the directory service to make this determination.

When redirecting the user's browser, the authenticator will set up to two header fields which cause, directly or indirectly, cookies to be stored on a users browser. The below table defines these header fields

Headers set by the Authenticator	
Name	**Value**
Location	URL to redirect to, either req.cgi.RET_URL or req.header.referrer
SetCookie	SESSION_TICKET_TXT=<session ticket>;secure
SetCookie	SESSION_TICKET_SIG=<digital signature of session ticket>

The first header, Location, is a standard HTTP header for redirecting browsers to a different URL. It is used in this instance to return the browser to the referring application.

The next two headers are SetCookie type headers, the standard HTTP header for setting a cookie on the user's browser. Since no domain or path is defined, this cookie will default to being returned only to the authenticator host, and will be returned to any URL on that host. The SESSION_TICKET_TXT contains the actual session ticket that states that the user has logged into the Intranet, while the SESSION_TICKET_SIG contains the digital signature verifying the authenticity of the ticket.

The authenticator also needs to set cookies named APP_TICKET_TXT and APP_TICKET_SIG, but cannot do so directly via the SetCookie header. The problem is that these two tickets must have their domain set to the application's domain, not the authenticator's domain. Since browsers can be configured to reject cookies set by a server on one domain for return to servers in another domain, an alternate method has been devised. A CGI variable, rmt_cookie, is added to the page when redirecting to the remote application. Since the remote application is WAP-enabled, it recognizes this CGI variable, and realizes that it must itself set a cookie on the user's browser, to be returned itself on successive hits by the user's browser.

CGI variables set by the Authenticator	
Name	**Value**
rmt_cookie	APP_TICKET_TXT=<plaintext application ticket>
rmt_cookie	APP_TICKET_SIG=<application ticket digital signature>

This cooperation between the WAP authenticator and WAP-enabled applications allows the authenticator to effectively set cross-domain cookies on users's browsers.

Currently, the only defined rmt_cookie values are APP_TICKET_TXT and APP_TICKET_SIG. Applications should ignore attempts to set other cookies via this mechanism.

5.2 Password Service

5.2.1 Overview

The password service provides a means of changing a user's password. The password service is just a basic WAP client application, but is covered in this protocol specification because its functionality is required for any authenticator service to be complete. WAP Server implementations MUST implement the password service. WAP Servers conform to the WAP client interface to implement the password service. See section 6.0 for details of the WAP client interface. Specifically, this means that the password service has an Intranet Application IID assigned to it.

The password service is designed to be accessed either through the HTML user interface using a HTTP form. The HTML interface can be customized on a per-site basis, but the form interface, including form variable names and allowable values, MUST conform to this specification.

5.2.2 Request Input Parameters

The URL for accessing the reference implementation password service is https://login.doc.gov/password.

The password service accepts requests only via HTTPS. The following CGI variables are processed. All other CGI variables are ignored.

CGI Variables processed by the Password Service	
Name	**Value**
OLD_PASSWORD	The user's current WAP password.
NEW_PASSWORD_1	The user's requested new WAP password.
NEW_PASSWORD_2	The user's requested new WAP password, as typed in a second time for confirmation.

The password service checks for the following cookies to determine if the user is authenticated for this application. Anonymous access is disallowed. The handling of these items is per section 6.

Cookies accepted by the Password Service	
Name	**Value**
APP_TICKET_TXT	Encrypted application Ticket
APP_TICKET_SIG	Digital signature of the application ticket

5.2.3 Execution

The server should first validate that the client is authenticated, as described in section 6.0.

For authenticated users, the WAP_CHG_PASSWORD page is presented. If the user successfully enters the old password and a valid new password, the WAP_OK_PASSWORD page should be presented.

If the OLD_PASSWORD field is correct, the WAP_BAD_PASSWORD page is presented, giving the user another chance to change the password.

If NEW_PASSWORD_1 does not match NEW_PASSWORD_2, then the WAP_TYPO_PASSWORD page is presented, giving the user another chance to change the password.

If the NEW_PASSWORD fields do not comply with site password requirements, then WAP_INVALID_PASSWORD page is returned, giving the user a chance to choose a different password.

In addition, servers MAY allow administrators to configure additional security features, such as a maximum number of retries before the account is locked out. These additional features can in no way interfere with the operation of the prescribed interface.

5.2.4 Response Output

Pages Returned by the Authenticator Password Service

WAP_CHG_PASSWORD page

- OLD_PASSWORD password input element
- NEW_PASSWORD_1 password input element
- NEW_PASSWORD_2 password input element
- submit button
- cancel button

WAP_TYPO_PASSWORD page

- Text explaining that passwords do not match
- OLD_ PASSWORD password input element
- NEW_PASSWORD_1 password input element

- NEW_PASSWORD_2 password input element
- submit button
- cancel button

WAP_BAD_PASSWORD page

- Text explaining that the old password is not correct
- OLD_ PASSWORD password input element
- NEW_PASSWORD_1 password input element
- NEW_PASSWORD_2 password input element
- submit button
- cancel button

WAP_INVALID_PASSWORD page

- Text explaining that the new password is insecure, and describing the rules in effect for passwords.
- OLD_ PASSWORD password input element
- NEW_PASSWORD_1 password input element
- NEW_PASSWORD_2 password input element
- submit button
- cancel button

WAP_OK_PASSWORD page

- Text indicating that password was changed successfully.

6 WAP-enabled Applications (WAP Client Interface)

6.1 Overview

The behavior a WAP-enabled application must exhibit is simpler than the behavior a WAP authenticator must exhibit. Basically, the WAP-enabled application, upon receiving a request for a resource that requires authentication, must check the incoming request for cookies named APP_TICKET_TXT and APP_TICKET_SIG. If these cookies are found, the APP_TICKET_SIG is verified against the APP_TICKET_TXT, using the public key of the authenticator. Successful comparison means that the ticket is authentic. The ticket is then checked for validity; USER_IP address is verified, APP_IID is verified, and RENEW time is verified. If all these pass, the user is authentic.

The first time an authenticator sends an application ticket to the application, it is not in the form of a cookie. Since the authenticator cannot set a cookie to be returned to the application, the authenticator instead sets a pair of CGI variables, both named ‘rmt_cookie’ (See section 5.1.4). One of the pair will be the TXT portion, and the other the SIG portion. Upon receiving a request with this CGI variable pair, the WAP-enabled application must verify the authenticity of the rmt_cookie by calculating a digital signature of the TXT portion with the authenticator’s public key, and compare it with the SIG portion. If the two compare correctly, the rmt_cookie is considered authentic. The application must redirect the browser back to very URL the browser is requesting, but with a SetCookie header set for the contents of both rmt_cookie variables (TXT and SIG portions) the application received.

Currently, only APP_TICKET_TXT and APP_TICKET_SIG are valid rmt_cookie entries. All others should be ignored.

If the WAP-enabled application gets a request without a APP_TICKET_TXT and APP_TICKET_SIG cookie pair, nor a rmt_cookie CGI variable pair containing APP_TICKET_TXT and APP_TICKET_SIG, or if he receives a ticket which has exceeded the renewal time, the application must set a CGI variable named APP_IID containing this application instance’s intranet id, and must set a CGI variable named RET_URL containing the URL the user should return to once authenticated, and must redirect the user’s browser to the authenticator’s ticket service URL.

6.2 Request Input Parameters

The WAP-enabled application accepts requests via HTTP or HTTPS.

No CGI variables are needed when requesting a resource from a WAP-enabled application.

The WAP-enabled application checks for the following cookie.

Cookies handled by WAP-enabled Applications	
Name	**Value**
APP_TICKET_TXT	The plaintext application ticket from the WAP authenticator.

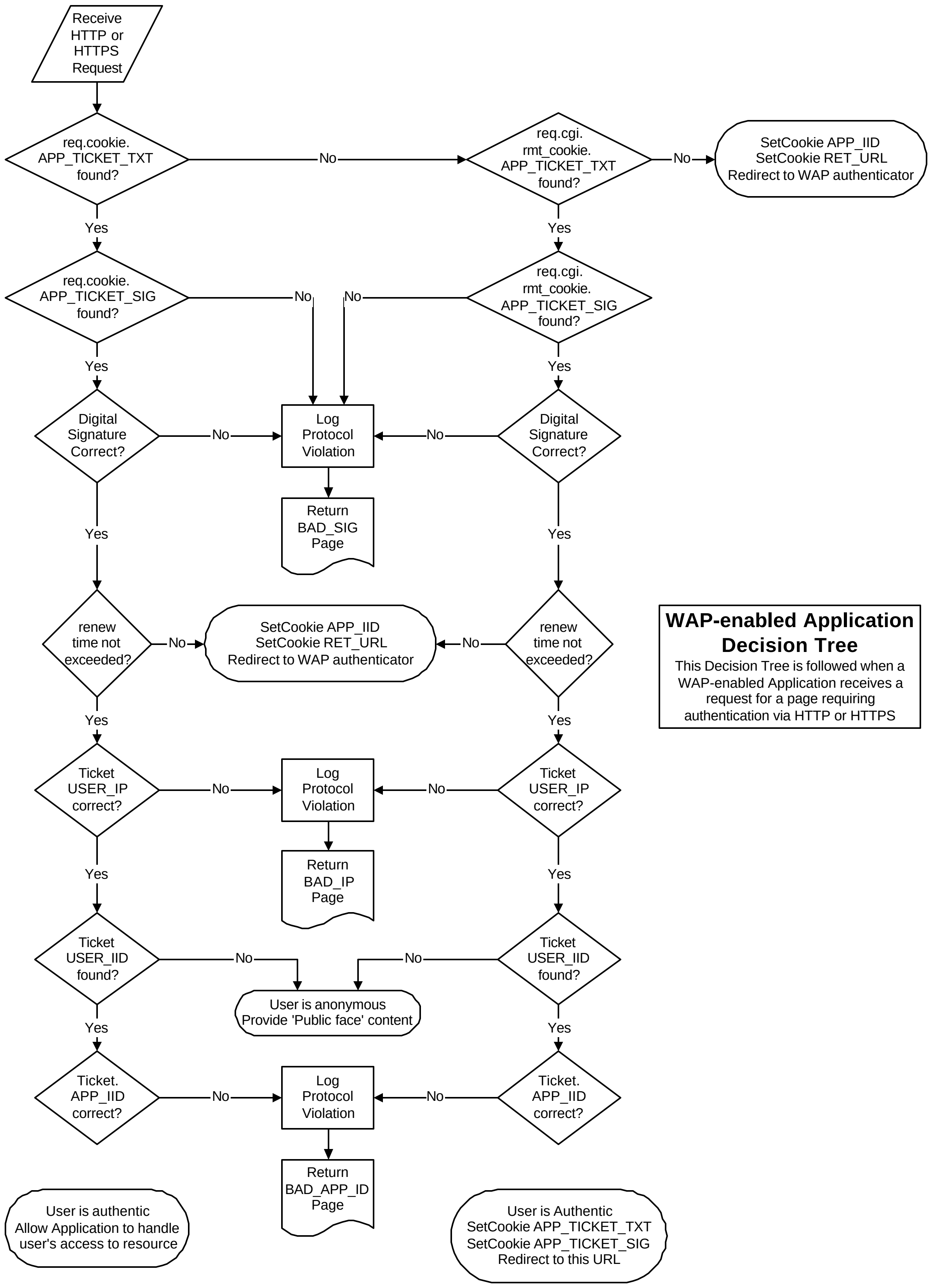
APP_TICKET_SIG	The digital signature of APP_TICKET_TXT, signed with the authenticator’s private key.

The WAP-enabled application checks for the following header.

HTTP CGI variables handled by WAP-enabled Applications	
Name	**Value**
rmt_cookie	APP_TICKET_TXT:<plaintext application ticket>
rmt_cookie	APP_TICKET_SIG:<application ticket digital signature>

6.3 Execution

The decision tree illustrated in the figure on the next page is followed each time a WAP-enabled application receives a request for a resource that requires authentication.


Receive HTTP or HTTPS Request
req.cookie. APP_TICKET_TXT found?
No
req.cgi. rmt_cookie. APP_TICKET_TXT found?
No
SetCookie APP_IID SetCookie RET_URL Redirect to WAP authenticator
Yes
Yes
req.cookie. APP_TICKET_SIG found?
No
No
req.cgi. rmt_cookie. APP_TICKET_SIG found?
Yes
Yes
Digital Signature Correct?
No
Log Protocol Violation
No
Digital Signature Correct?
Return BAD_SIG Page
Yes
Yes
renew time not exceeded?
No
SetCookie APP_IID SetCookie RET_URL Redirect to WAP authenticator
No
renew time not exceeded?
Yes
Yes
Ticket USER_IP correct?
No
Log Protocol Violation
No
Ticket USER_IP correct?
Return BAD_IP Page
Yes
Yes
Ticket USER_IID found?
No
No
Ticket USER_IID found?
User is anonymous Provide 'Public face' content
Yes
Yes
Ticket. APP_IID correct?
No
Log Protocol Violation
No
Ticket. APP_IID correct?
Return BAD_APP_ID Page
User is authentic Allow Application to handle user's access to resource
User is Authentic SetCookie APP_TICKET_TXT SetCookie APP_TICKET_SIG Redirect to this URL

WAP-enabled Application Decision Tree

This Decision Tree is followed when a WAP-enabled Application receives a request for a page requiring authentication via HTTP or HTTPS

Commentary on WAP-enabled Application Decision Tree

Receive HTTPS request	This is the starting point for all transactions. A HTTP or HTTPS connection is made for a resource requiring user authentication.
req.cookie.APP_TICKET_TXT found?	Check to see if the user's browser has submitted a cookie named APP_TICKET_TXT. This should contain a plaintext application ticket.
req.cookie.APP_TICKET_SIG found?	Check to see if the user's browser has submitted a cookie named APP_TICKET_SIG. This should contain a digital signature.
req.cgi.rmt_cookie.APP_TICKET_TXT found?	Check to see if the request contains a CGI variable of the form rmt_cookie=APP_TICKET_TXT… This is a request for the application to set a cookie, supposedly on behalf of the authenticator.
req.cgi.rmt_cookie.APP_TICKET_SIG found?	Check to see if the request contains a CGI variable of the form rmt_cookie=APP_TICKET_SIG… This should be a digital signature of APP_TICKET_TXT.
Digital Signature Correct?	Use the authenticator's public key to verify that APP_TICKET_SIG is in fact a valid digital signature of APP_TICKET_TXT. If it is, then the ticket is authentic.
renew time not exceeded?	Check to verify that the 'expire time' contained in APP_TICKET_TXT has not been exceeded. If it has, the application needs to obtain a new ticket by redirecting the user to the authenticator.
Ticket USER_IP correct?	Verify that the user's browser is in fact connecting from the address listed as USER_IP in the ticket.
Ticket USER_IID found?	Check to see if a USER_IID is provided with the ticket. If not, then the user is connecting in anonymous mode.
Ticket APP_IID correct?	Verify that the ticket contains this application's Intranet ID.
Log Protocol Violation	An invalid ticket has been received. All such attempts should be logged so that administrators can take appropriate action.

Return BAD_SIG Page	A page should be returned to give the user further guidance on appropriate actions if the APP_TICKET_SIG is not genuine. The contents will depend on site preferences – users could be notified that the ticket was rejected, or a generic 'Login Failed' message could be displayed.
Return BAD_IP Page	A page should be returned to give the user further guidance on appropriate actions if the APP_TICKET_TXT lists a USER_IP address that does not match the IP the browser is currently connecting from.
User is anonymous. Return 'public face' content.	The user is not authenticated. The application may, at its discretion, provide a 'public face' interface to the anonymous user.
SetCookie APP_IID SetCookie RET_URL Redirect to WAP authenticator	If there is no application ticket, or if there is a session ticket requiring renewal, redirect the user's browser to the authenticator. Set RET_URL to the URL the authenticator should return the user to, and APP_ID to this application's IID.
User is authentic. SetCookie APP_TICKET_TXT SetCookie APP_TICKET_SIG Redirect to this URL	The user is authentic, and the application needs to persistently store the application ticket and signature the user presented in a cookie on the user's browser. Set cookies for APP_TICKET_TXT and APP_TICKET_SIG, and redirect the user back to this URL.
User is authentic. Allow application to handle user's access to resource	The user is authenticated. It is now up to the application to decide what, if any, content the application wishes to provide the user.

6.4 Outputs

Pages Returned by the WAP-enabled Application

The WAP client API will provide default pages for the following. Applications are encouraged to override this default behavior, and provide their own versions of the following pages.

BAD_SIG page

- Message informing user that the SESSION_COOKIE submitted failed the digital signature test.

BAD_APP_ID page

- Message informing user that SESSION_COOKIE is invalid because it is not for the application he intends to connect to.

BAD_IP page

- Message informing user that their session is invalid because their IP address has changed.

Redirects returned by the WAP-enabled Application

Redirect to WAP authenticator

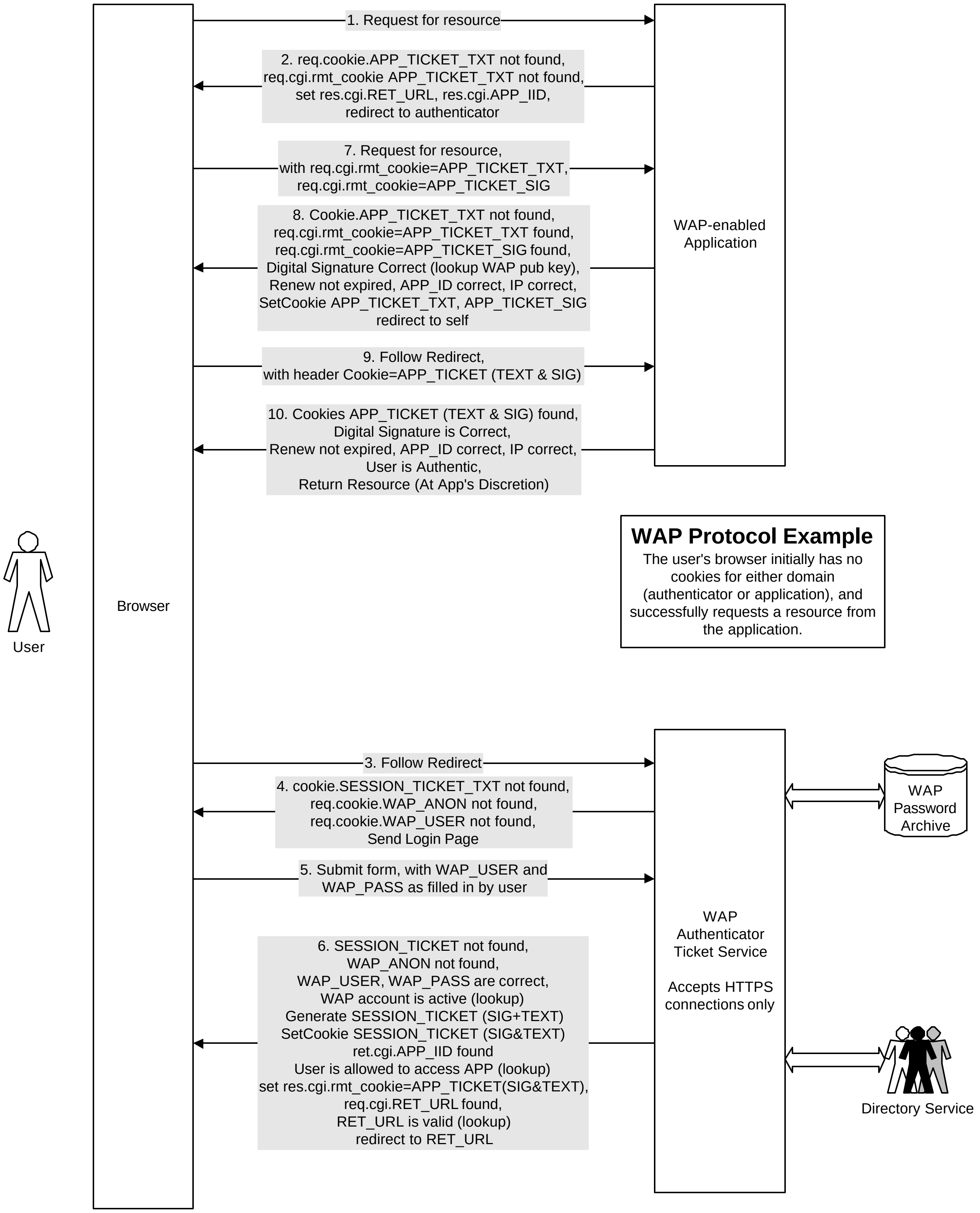
- Redirect to the WAP authenticator URL after setting CGI variables APP_IID and RET_URL.

Redirect to this URL

- Redirect to one's own URL after setting APP_TICKET_TXT and APP_TICKET_SIG cookies.

7 Scenarios

User requesting an application; user not authenticated. (See figure on the next page.)


1. Request for resource
2. req.cookie.APP_TICKET_TXT not found, req.cgi.rmt_cookie APP_TICKET_TXT not found, set res.cgi.RET_URL, res.cgi.APP_IID, redirect to authenticator
7. Request for resource, with req.cgi.rmt_cookie=APP_TICKET_TXT, req.cgi.rmt_cookie=APP_TICKET_SIG
8. Cookie.APP_TICKET_TXT not found, req.cgi.rmt_cookie=APP_TICKET_TXT found, req.cgi.rmt_cookie=APP_TICKET_SIG found, Digital Signature Correct (lookup WAP pub key), Renew not expired, APP_ID correct, IP correct, SetCookie APP_TICKET_TXT, APP_TICKET_SIG redirect to self
9. Follow Redirect, with header Cookie=APP_TICKET (TEXT & SIG)
10. Cookies APP_TICKET (TEXT & SIG) found, Digital Signature is Correct, Renew not expired, APP_ID correct, IP correct, User is Authentic, Return Resource (At App's Discretion)
WAP-enabled Application
User
Browser
3. Follow Redirect
4. cookie.SESSION_TICKET_TXT not found, req.cookie.WAP_ANON not found, req.cookie.WAP_USER not found, Send Login Page
5. Submit form, with WAP_USER and WAP_PASS as filled in by user
6. SESSION_TICKET not found, WAP_ANON not found, WAP_USER, WAP_PASS are correct, WAP account is active (lookup) Generate SESSION_TICKET (SIG+TEXT) SetCookie SESSION_TICKET (SIG&TEXT) ret.cgi.APP_IID found User is allowed to access APP (lookup) set res.cgi.rmt_cookie=APP_TICKET(SIG&TEXT), req.cgi.RET_URL found, RET_URL is valid (lookup) redirect to RET_URL
WAP Authenticator Ticket Service
Accepts HTTPS connections only
WAP Password Archive
Directory Service

WAP Protocol Example

The user's browser initially has no cookies for either domain (authenticator or application), and successfully requests a resource from the application.

8 Encryption and Digital Signatures

8.1 Digital Signatures

All digital signatures will be per FIPS Pub 186.2 (See [FIPS186-2]. The reference implementation will use RSA encryption using network byte order.

See [CRYPTO] for details.

9 Unresolved Issues

Browsers Using IP Address Pools

Need to determine if ISPs still use pools of IP addresses for web browsers, thus allowing a browser's IP address to change during a session. If this can occur, the protocol must allow for users to re-login and have the current block (or range) recorded as valid.

Applications saving state of user

Need to determine how to push POST style CGI variables back via a redirect. Some applications may only support POST operations so URL-encoding the redirect may not work for some legacy applications.

10 References

[RFC2109] HTTP State Management Mechanism (RFC 2109), available http://www.faqs.org/rfcs/rfc2109.html

[RFC2119] Key words for use in RFCs to Indicate Requirement Levels (RFC 2119), available http://www.faqs.org/rfcs/rfc2119.html

[RFC2616] Hypertext Transfer Protocol -- HTTP/1.1 (RFC 2616), available http://www.faqs.org/rfcs/rfc2616.html

[SSL] The SSL Protocol, Version 3.0, available http://home.netscape.com/eng/ssl3/ssl-toc.html

[CRYPTO] Department of Commerce Intranet Cryptography Support API & Tutorial

[FIPS186-2] FIPS Pub 186-2: Digital Signature Standard, available http://csrc.nist.gov/fips/fips186-2.pdf

Appendix B: WAP Client API for the DOC Intranet

Version 1.0

Revised: 2000.11.12

1 Introduction

Applications utilizing the WAP protocol for authentication within the DOC intranet require a standard toolset to facilitate executing the client protocol. The purpose of the document is to provide API's enabling client applications to easily authenticate users through use of the WAP protocol. This document describes the API for Java, Perl, and PHP.

2 Terminology

API

> Application Programming Interface. A set of routines provided for a developer to access programmatic functionality that has already been developed.

DOC

> Department of Commerce.

IID

> Intranet ID. The identifier for any entry in the DOC intranet directory service. The
>
> IID is unique in the entire directory service. For instance, an application cannot have the same IID as a person. It is a string composed of numbers and upper and lower case letters, and is case-sensitive.

3 Overview

The Directory API provides applications an easy means of accessing DOC directory service and Group Service information. Rather than making calls to the LDAP server and group server directly, the application uses API calls that deal with networking, protocol, and data handling issues. The programmer need only concentrate on the application they are trying to build.

A diagram showing the Directory API’s place in the directory portion of the DOC intranet infrastructure is included in *Figure 1* in *Appendix D: Directory Schema for the DOC Intranet Directory Service*.

4 The Directory API for Java

This class manages WAP authentication. Applications usually handle requests as follows:

```
AuthenticationManager auth = new AuthenticationManager(...);

try {

      auth.authenticate();

      if (auth.getPayload() != null)

         // restore state from payload using getPayload

      if (auth.isAnonymous()) {

         // Provide anonymous service

      } else if (auth.isAlternateUsernameProvided()) {

         // Provide service using getAuthAlternateUsername

         // as user identifier

      } else {

         // Provide service using getAuthUserIID as user identifier

      }

   } catch (ProtocolViolationException pve) {

      auth.sendErrorPage(...);

      return;

   } catch (RedirectRequiredException rre) {

      auth.setPayload (...);

      auth.sendRedirect();

      return;

   }
```

4.1 The AuthenticationManager Class

Signature

public class **AuthenticationManager** extends Object

Variables

public static final int REDIR_NONE	WAP does not require redirection.
public static final int REDIR_AUTH	WAP requires redirection to Authenticator
public static final int REDIR_SELF	WAP requires redirection to self in order to set cookies on client.
public static final int PAGE_BAD_SIG	WAP BAD_SIG page handle.
public static final int PAGE_BAD_APP_IID	WAP BAD_APP_IID page handle.
public static final int PAGE_BAD_IP	WAP BAD_IP page handle.

Constructors

AuthenticationManager

```
public AuthenticationManager(String myApplicationIID,
                             URL myReturnURL,
                             docDirectory dirSvc,
                             HttpServletRequest req,
                             HttpServletResponse res,
                             TicketService tktSvc)
```

Defines an authenticator with an initial Request/Response pair.

Parameters:

myApplicationIID - The Intranet ID of the calling application.

myReturnURL - The URL to which clients should return if they are redirected elsewhere.

dirSvc - The DOC Directory Server reference as a docDirectory object.

req - The incoming servlet request.

res - The outgoing servlet response.

tktSvc - The TicketService object describing the WAP ticket service to use.

Methods

---

setAuditLog

public void setAuditLog(OutputStream logStream)

Define the stream to which logging should be directed. The logging is for protocol and security violations.

Parameters:

logStream - the output stream to direct audit logging to.

---

setPayload

public void setPayload(String payload)

Define a WAP payload. The WAP payload can be used to store state when redirecting a request to the authenticator. The authenticator is required to maintain the payload, and to return it when redirecting the user back to the calling application. It is returned as a CGI GET parameter named payload.

Parameters:

payload - The string representation of a payload.

---

getPayload

public String getPayload()

If a payload is present in the incoming request, this returns the String representation of the payload, else null. # @return String representation of a payload, null if request has no payload

---

sendRedirect

public void sendRedirect()

Modifies the response object to set a return code of 301 and add a location header for the purpose of redirection, and commits the response object to the servlet output stream. This is used to redirect a user who either lacks authentication information or who has authentication information requiring renewal to the Ticket Service URL, or to redirect the user back to this URL, setting cookies on the user's browser in the process.

---

authenticate

public void authenticate() throws ProtocolViolationException, RedirectRequiredException

Examines the incoming request for authentication ticket, and checks any ticket present for validity. AuthenticationManager objects go through two phases: first they are created, then they authenticate the request. Operations that depend on authentication having been tested, like getAuthUserIID, will implicitly call authenticate if necessary.

Throws: ProtocolViolationException

if the WAP protocol is violated. This happens when client IP address does not match that on ticket, ticket fails digital signature test, or ticket's Application IID does not match the calling application's IID. The specific subclass indicates which violation occurred.

Throws: RedirectRequiredException

authentication cannot complete without a redirect as required by WAP. The specific subclass indicates which redirection case is needed.

---

isRedirectRequired

public boolean isRedirectRequired()

Checks if the WAP protocol requires a redirection for this connection. Redirections are required if connecting clients have no tickets, or if they have a old ticket requiring renewal, or if the authenticator has set CGI variables indicating the application should set cookies on behalf of the authenticator. Equivalent to (getRedirectType() != REDIR_NONE).

Returns:

whether redirect is required to complete authentication protocol

See Also:

sendRedirect, getRedirectURL, getRedirectType

---

isRenewRequired

public boolean isRenewRequired()

Checks if the WAP protocol requires a renewal request to be sent to the authenticator. Renewals are required if the connecting client has a ticket with an expired renew date.

Returns:

whether application ticket is outdated and requires renewal

See Also:

sendRedirect

---

isValidTicketProvided

public boolean isValidTicketProvided()

Checks if the connecting client provided a valid, up to date application ticket. This method checks the following:

- Posession of a ticket
- Valid digital signature by the authenticator
- Renew time has not expired
- Client IP address matches that on ticket

Returns:

whether client submitted a valid application ticket via a cookie

---

isAnonymous

public boolean isAnonymous()

Checks if the user is accessing this application anonymously. This happens when a user declines to provide authentication information, or when the authenticator is configured to automatically return anonymous access to this application.

Returns:

whether the user is connected anonymously (has application ticket but no UserIID)

---

isAlternateUsernameProvided

public boolean isAlternateUsernameProvided()

Checks if the client's ticket contains an alternate username. Alternate usernames are provided on an application-specific basis by the authenticator to legacy applications that do not understand Intranet ID's, and require instead a different username be used by the user.

Returns:

whether the authenticator provided an alternate username for this application to use instead of UserIID

---

getAuthAlternateUsername

public String getAuthAlternateUsername()

Returns the Authenticated Alternate Username if so included in the authentication ticket. Returns null if no alternate username is provided.

Returns:

the alternate username provided by the authenticator, or null if none was provided

See Also:

isAlternateUsernameProvided

---

getAuthUserIID

public String getAuthUserIID()

Returns the Intranet ID (IID) of the authenticated user, or null for anonymous or unauthenticated users.

Returns:

the authenticated user IID of the client

---

getRedirectURL

public URL getRedirectURL()

If WAP requires a redirect, this call returns the URL to be redirected to, else null.

Returns:

the URL which `sendRedirect` will redirect the client to, or null if no redirection is required.

---

getRedirectType

public int getRedirectType()

Returns REDIR_AUTH if WAP requires redirect to authenticator, REDIR_SELF if WAP requires redirect to self, and REDIR_NONE if WAP does not require a redirect.

Returns:

one of REDIR_AUTH, REDIR_SELF or REDIR_NONE, reflecting what type of redirect is required by the WAP protocol.

---

sendErrorPage

public void sendErrorPage(int pageID)

Sends a fixed page out in accordance with WAP protocol.

Parameters:

pageID - One of PAGE_BAD_SIG, PAGE_BAD_IP or PAGE_BAD_APP_IID

4.2 The BadAppIIDException Class

Signature

public class **BadAppIIDException** extends ProtocolViolationException

Constructors

BadAppIIDException

public BadAppIIDException()

Constructs a BadAppIIDException with no specified detail message.

BadAppIIDException

public BadAppIIDException(String s)

Constructs an BadAppIIDException with the specified detail message.

Parameters:

s - the detail message

4.3 The BadClientIPAddressException Class

Signature

public class **BadClientIPAddressException** extends ProtocolViolationException

Constructors

BadClientIPAddressException

```
public BadClientIPAddressException()
```

Constructs a BadClientIPAddressException with no specified detail message.

BadClientIPAddressException

```
public BadClientIPAddressException(String s)
```

Constructs an BadClientIPAddressException with the specified detail message.

Parameters:

s - the detail message

4.4 The BadSignatureException Class

Signature

public class **BadSignatureException** extends ProtocolViolationException

Constructors

BadSignatureException

```
public BadSignatureException()
```

Constructs a BadSignatureException with no specified detail message.

BadSignatureException

```
public BadSignatureException(String s)
```

Constructs an BadSignatureException with the specified detail message.

Parameters:

s - the detail message

4.5 The ProtocolViolationException Class

Signature

public class **ProtocolViolationException** extends Exception

Constructors

ProtocolViolationException

public ProtocolViolationException()

Constructs a ProtocolViolationException with no specified detail message.

ProtocolViolationException

public ProtocolViolationException(String s)

Constructs an Exception with the specified detail message.

Parameters:

s - the detail message

4.6 The RedirectNoTicketException Class

Signature

public class **RedirectNoTicketException** extends RedirectRequiredException

Constructors

RedirectNoTicketException

public RedirectNoTicketException()

Constructs a RedirectNoTicketException with no specified detail message.

RedirectNoTicketException

public RedirectNoTicketException(String s)

Constructs an RedirectNoTicketException with the specified detail message.

Parameters:

s - the detail message

4.7 The RedirectRenewalException Class

Signature

public class **RedirectRenewalException** extends RedirectRequiredException

Constructors

RedirectRenewalException

public RedirectRenewalException()

Constructs a RedirectRenewalException with no specified detail message.

RedirectRenewalException

public RedirectRenewalException(String s)

Constructs an RedirectRenewalException with the specified detail message.

Parameters:

s - the detail message

4.8 The RedirectRequiredException Class

Signature

public class **RedirectRequiredException** extends Exception

Constructors

RedirectRequiredException

public RedirectRequiredException()

Constructs a RedirectRequiredException with no specified detail message.

RedirectRequiredException

public RedirectRequiredException(String s)

Constructs an RedirectRequiredException with the specified detail message.

Parameters:

s - the detail message

4.9 The RedirectToSelfException Class

Signature

public class **RedirectToSelfException** extends RedirectRequiredException

Constructors

RedirectToSelfException

```
public RedirectToSelfException()
```

Constructs a RedirectToSelfException with no specified detail message.

RedirectToSelfException

```
public RedirectToSelfException(String s)
```

Constructs an RedirectToSelfException with with the specified detail message.

Parameters:

s - the detail message

4.10 The TicketService Class

Signature

public class **TicketService** extends Object

Constructors

TicketService

```
public TicketService(URL ticketServiceURL,
                     byte ticketServicePublicKey[])
```

Creates a ticket service object to hold information defining a WAP authenticator ticket service

Parameters:

ticketServiceURL - the well-known URL of the ticket service ticketServicePublicKey - the public key used to authenticate the authenticator

---

TicketService

```
public TicketService()
```

Creates an uninitialized ticket service object

Methods

---

setURL

```
public void setURL(URL URL)
```

Sets or replaces the URL representing the well-known URL of the ticket service.

Parameters:

URL - the well-known ticket service URL

---

setPublicKey

```
public void setPublicKey(byte publicKey[])
```

Sets or replaces the public key used to authenticate with this ticket service.

Parameters:

publicKey - the well-known ticket service public key

---

getURL

```
public URL getURL()
```

Return the well-known URL of the ticket service or null if not set.

---

getPublicKey

```
public byte[] getPublicKey()
```

Return the well-known ticket service public-key, or null if not set.

---

5 The WAP Client API for Perl

The Perl client API's reside in a module named DOCIntranet.pm.   They require CPAN modules Exception and URI:URL. The main class is the AuthenticationManager class, which requires a TicketService class and a DocDirectory class in its constructor. Various exceptions round out the provided set of classes.

Applications usually handle requests as follows:

```
use Exception;

use URI::URL;

use DOCIntranet;


$auth = new AuthenticationManager(...);


try {

   $auth->authenticate();

   if ($auth->getPayload() != null)

      // restore state from payload using getPayload

   if ($auth->isAnonymous()) {

      // Provide anonymous service

   } else if ($auth->isAlternateUsernameProvided()) {

      // Provide service using getAuthAlternateUsername

      // as user identifier

   } else {

      // Provide service using getAuthUserIID as user identifier

   }

}


catch DOCIntranet::ProtocolViolationException pve =>
```

```
      sub{
          $auth.sendErrorPage(...);
          return;
       },
    catch DOCIntranet::RedirectRequiredException rre =>
      sub{
          $auth.setPayload (...);
          $auth.sendRedirect();
          return;
       }
    }
```

5.1 The AuthenticationManager Class

Signature

package **AuthenticationManager;**

@ISA = (‘UNIVERSAL’);

Variables

REDIR_NONE	WAP does not require redirection.
REDIR_AUTH	WAP requires redirection to Authenticator
REDIR_SELF	WAP requires redirection to self in order to set cookies on client.
PAGE_BAD_SIG	WAP BAD_SIG page handle.
PAGE_BAD_APP_IID	WAP BAD_APP_IID page handle.
PAGE_BAD_IP	WAP BAD_IP page handle.

Constructors

```
$auth=new AuthenticationManager (
          myApplicationIID,
          myReturnURL,
          dirSvc,
          tktSvc )
```

Parameters:

myApplicationIID - The Intranet ID of the calling application.

myReturnURL - The URI::URL to which clients should return if they

are redirected elsewhere.

dirSvc - The DOC Directory Server reference as a docDirectory object.

tktSvc - The TicketService object describing the WAP ticket service to use.

Methods

---

setAuditLog

`$auth->setAuditLog(logFile)`

Define the stream to which logging should be directed. The logging is for protocol and security violations.

Parameters:

logFile - the output file to direct audit logging to.

---

setPayload

`$auth->setPayload(payload)`

Define a WAP payload. The WAP payload can be used to store state when redirecting a request to the authenticator. The authenticator is required to maintain the payload, and to return it when redirecting the user back to the calling application. It is returned as a CGI GET parameter named payload.

Parameters:

payload - The string representation of a payload.

getPayload

`$auth->getPayload()`

If a payload is present in the incoming request, this returns the String representation of the payload, else null.

Returns:

String representation of a payload, null if request has no

payload

sendRedirect

`$auth->sendRedirect()`

Outputs a redirection HTTP response as required by the WAP protocol. This is used to redirect a user who either lacks authentication information or who has authentication information requiring renewal to the Ticket Service URL, or to redirect the user back to this URL, setting cookies on the user's browser in the process.

authenticate

`$auth->authenticate()`

Examines the incoming request for authentication ticket, and checks any ticket present for validity. AuthenticationManager objects go through two phases: first they are created, then they authenticate the request. Operations that depend on authentication having been tested, like getAuthUserIID, will implicitly call authenticate if necessary.

Throws: ProtocolViolationException

if the WAP protocol is violated. This happens when client IP address does not match that on ticket, ticket fails digital signature test, or ticket's Application IID does not match the calling application's IID. The specific subclass indicates which violation occurred.

Throws: RedirectRequiredException

authentication cannot complete without a redirect as required by WAP. The specific subclass indicates which redirection case is needed.

isRedirectRequired

`$auth->isRedirectRequired()`

Checks if the WAP protocol requires a redirection for this connection. Redirections are required if connecting clients have no tickets, or if they have a old ticket requiring renewal, or if the authenticator has set CGI variables indicating the application should set cookies on behalf of the authenticator. Equivalent to (getRedirectType() != REDIR_NONE).

Returns:

boolean indicating whether redirect is required to complete authentication protocol

See Also:

sendRedirect, getRedirectURL, getRedirectType

---

isRenewRequired

`$auth->`**`isRenewRequired`**`()`

Checks if the WAP protocol requires a renewal request to be sent to the authenticator. Renewals are required if the connecting client has a ticket with an expired renew date.

Returns:

boolean indicating whether application ticket is outdated and requires renewal

See Also:

sendRedirect

---

isValidTicketProvided

`$auth->`**`isValidTicketProvided`**`()`

Checks if the connecting client provided a valid, up to date application ticket. This method checks the following:

- Posession of a ticket
- Valid digital signature by the authenticator
- Renew time has not expired
- Client IP address matches that on ticket

Returns:

boolean indicating whether client submitted a valid application ticket via a cookie

---

isAnonymous

`$auth->`**`isAnonymous`**`()`

Checks if the user is accessing this application anonymously. This happens when a user declines to provide authentication information, or when the authenticator is configured to automatically return anonymous access to this application.

Returns:

boolean indicating whether the user is connected anonymously (has application ticket but no UserIID)

---

isAlternateUsernameProvided

`$auth->`**`isAlternateUsernameProvided`**`()`

Checks if the client's ticket contains an alternate username. Alternate usernames are provided on an application-specific basis by the authenticator to legacy applications which do not understand Intranet ID's, and require instead a different username be used by the user.

Returns:

boolean indicating whether the authenticator provided an alternate username for this application to use instead of UserIID

---

getAuthAlternateUsername

`$auth->`**`getAuthAlternateUsername`**`()`

Returns the Authenticated Alternate Username if so included in the authentication ticket. Returns null if no alternate username is provided.

Returns:

the alternate username string provided by the authenticator, or **undef** if none was provided

See Also:

isAlternateUsernameProvided

---

getAuthUserIID

`$auth->`**`getAuthUserIID`**`()`

Returns the Intranet ID (IID) of the authenticated user, or **undef** for anonymous or unauthenticated users.

Returns:

the authenticated user IID string of the client

getRedirectURL

`$auth``->`**`getRedirectURL`**`()`

If WAP requires a redirect, this call returns the URI::URL to be redirected to, else it return **undef**.

Returns:

the URI::URL which `sendRedirect` will redirect the client to, or **undef** if no redirection is required.

---

getRedirectType

`$auth``->`**`getRedirectType`**`()`

Returns REDIR_AUTH if WAP requires redirect to authenticator, REDIR_SELF if WAP requires redirect to self, and REDIR_NONE if WAP does not require a redirect.

Returns:

one of the defined integers REDIR_AUTH, REDIR_SELF or REDIR_NONE, reflecting what type of redirect is required by the WAP protocol.

---

sendErrorPage

`$auth``->`**`sendErrorPage`**`(`*`pageID`*`)`

Sends a fixed page out in accordance with WAP protocol.

Parameters:

pageID - One of the defined integers PAGE_BAD_SIG, PAGE_BAD_IP or PAGE_BAD_APP_IID

5.2 The BadAppIIDException Class

Signature

package **BadAppIIDException;**

@ISA = (‘ProtocolViolationException’);

Constructors

BadAppIIDException

`$exc = new BadAppIIDException()`

Constructs a BadAppIIDException.

5.3 The BadClientIPAddressException Class

Signature

package **BadClientIPAddressException;**

@ISA = (‘ProtocolViolationException’);

Constructors

BadClientIPAddressException

`$exc = new BadClientIPAddressException ()`

Constructs a BadClientIPAddressException.

5.4 The BadSignatureException Class

Signature

package **BadSignatureException;**

@ISA = (‘ProtocolViolationException’);

Constructors

BadSignatureException

`$exc = new BadSignatureException ()`

Constructs a BadSignatureException.

5.5 The ProtocolViolationException Class

Signature

package **ProtocolViolationException;**

@ISA = (‘Exception’);

Constructors

ProtocolViolationException

$exc = **new ProtocolViolationException** ()

Constructs a ProtocolViolationException.

5.6 The RedirectNoTicketException Class

Signature

package **RedirectNoTicketException;**

@ISA = (‘RedirectRequiredException);

Constructors

RedirectNoTicketException

$exc = **new RedirectNoTicketException** ()

Constructs a RedirectNoTicketException.

5.7 The RedirectRenewalException Class

Signature

package **RedirectRenewalException;**

@ISA = (‘RedirectRequiredException);

Constructors

RedirectRenewalException

$exc = **new RedirectRenewalException** ()

Constructs a RedirectRenewalException.

5.8 The RedirectRequiredException Class

Signature

package **RedirectRequiredException;**

@ISA = (‘Exception);

Constructors

RedirectRequiredException

`$exc = new RedirectRequiredException ()`

Constructs a RedirectRequiredException.

5.9 The RedirectToSelfException Class

Signature

package **RedirectToSelfException;**

@ISA = (‘RenewalRequiredException);

Constructors

RedirectToSelfException

`$exc = new RedirectToSelfException ()`

Constructs a RedirectToSelfException.

5.10 The TicketService Class

Signature

package **TicketService;**

@ISA = (‘UNIVERSAL’);

Constructors

TicketService

`$auth=new TicketService(`

ticketServiceURL,

ticketServicePublicKey)

Creates a ticket service object to hold information defining a WAP authenticator ticket service

Parameters:

ticketServiceURL - the well-known URI::URL of the ticket service

ticketServicePublicKey - the public key used to authenticate the authenticator

---

TicketService

`$auth = new TicketService()`

Creates an uninitialized ticket service object

Methods

---

setURL

`$auth->setURL(URL)`

Sets or replaces the URI::URL representing the well-known URL of the ticket service.

Parameters:

URL - the well-known ticket service URI::URL

---

setPublicKey

`$auth->setPublicKey(publicKey)`

Sets or replaces the public key used to authenticate with this ticket service.

Parameters:

publicKey - the well-known ticket service public key string

---

getURL

`$auth->getURL()`

Return the well-known URI::URL of the ticket service or **undef** if not set.

---

getPublicKey

$auth->**getPublicKey**()

Return the well-known ticket service public-key, or **undef** if not set.

6 The WAP Client API for PHP

The PHP client API's reside in a library file named DOCIntranet.inc.   The main class is the AuthenticationManager class, which requires a TicketService class and a DocDirectory class in its constructor.

Applications usually handle requests as follows:

```
include "DOCIntranet.inc";


$auth = new AuthenticationManager(...);


$rv = $auth->authenticate();


isSet($rv) or switch ($doc_errno) {
  case 0:          // not really an error
       break;
  case BAD_APP_IID:
  case BAD_CLIENT_IP:
  case BAD_SIG:
       $auth.sendErrorPage($rv);        // does not return
       break;
  case REDIRECT_AUTH:
  case REDIRECT_SELF
       $auth.setPayload(...);
       $auth.sendRedirect();      // does not return
       break;
}


if ($auth->getPayload())
```

```
        // restore state from payload using getPayload
    if ($auth->isAnonymous()) {
        // Provide anonymous service
    } else if ($auth->isAlternateUsernameProvided()) {
        // Provide service using getAuthAlternateUsername
        // as user identifier
    } else {
        // Provide service using getAuthUserIID as user identifier
    }
```

6.1 The AuthenticationManager Class

Signature

class **AuthenticationManager;**

Variables

int doc_errno	error number of last API call, or 0 on success

Defined Constants

int REDIR_NONE	WAP does not require redirection.
int REDIR_AUTH	WAP requires redirection to Authenticator
int REDIR_SELF	WAP requires redirection to self in order to set cookies on client.
int BAD_SIG	WAP BAD_SIG page handle.
int BAD_APP_IID	WAP BAD_APP_IID page handle.
int BAD_CLIENT_IP	WAP BAD_IP page handle.

Constructors

```
$auth=new AuthenticationManager (
          string myApplicationIID,
          string myReturnURL,
          DocDirectory dirSvc,
          TicketService tktSvc )
```

Parameters:

myApplicationIID - The Intranet ID of the calling application.

myReturnURL - The URI::URL to which clients should return if they

are redirected elsewhere.

dirSvc - The DOC Directory Server reference as a docDirectory object.

tktSvc - The TicketService object describing the WAP ticket service to use.

Methods

setAuditLog

```
void $auth->setAuditLog(string logFile)
```

Define the file to which logging should be directed. The logging is for protocol and security violations.

Parameters:

logFile - the output file to direct audit logging to.

setPayload

```
void $auth->setPayload(string payload)
```

Define a WAP payload. The WAP payload can be used to store state when redirecting a request to the authenticator. The authenticator is required to maintain the payload, and to return it when redirecting the user back to the calling application. It is returned as a CGI GET parameter named payload.

Parameters:

payload - The string representation of a payload.

getPayload

string *$auth*->**getPayload**(void)

If a payload is present in the incoming request, this returns the String representation of the payload, else returns an unbound value and sets $doc_errno.

Returns:

String representation of a payload, an unbound value if request has no

payload. On errors, also sets `$doc_errno`.

sendRedirect

`void` ***$auth*->sendRedirect**`(void)`

Outputs a redirection HTTP response as required by the WAP protocol. This is used to redirect a user who either lacks authentication information or who has authentication information requiring renewal to the Ticket Service URL, or to redirect the user back to this URL, setting cookies on the user's browser in the process.

Return:

This function does not return control to the calling process.

authenticate

int *$auth*->**authenticate**(void)

Examines the incoming request for authentication ticket, and checks any ticket present for validity. AuthenticationManager objects go through two phases: first they are created, then they authenticate the request. Operations that depend on authentication having been tested, like getAuthUserIID, will implicitly call authenticate if necessary.

Returns:

On error, returns an unbound value, and sets `$doc_errno` to one of the following values.

```
BAD_CLIENT_IP when the client IP address does not match
               the IP addrres in the presented ticket
BAD_APP_IID        when the application IID does not match the
                        application IID in the ticket
```

`BAD_SIG`	`when the ticket fails the digital signature comparison test`
`REDIRECT_AUTH`	`when a redirct to the authenticator is required`
`REDIRECT_SELF`	`when a redirect back to the client is required in order to set cookies`

isRedirectRequired

`bool` ***`$auth`***`->`**`isRedirectRequired`**`(void)`

Checks if the WAP protocol requires a redirection for this connection. Redirections are required if connecting clients have no tickets, or if they have a old ticket requiring renewal, or if the authenticator has set CGI variables indicating the application should set cookies on behalf of the authenticator. Equivalent to (getRedirectType() != REDIR_NONE).

Returns:

boolean indicating whether redirect is required to complete authentication protocol , or an unbound value if `authenticate` was called and generated an error.

See Also:

sendRedirect, getRedirectURL, getRedirectType, authenticate

isRenewRequired

`bool` ***`$auth`***`->`**`isRenewRequired`**`(void)`

Checks if the WAP protocol requires a renewal request to be sent to the authenticator. Renewals are required if the connecting client has a ticket with an expired renew date.

Returns:

boolean indicating whether application ticket is outdated and requires renewal , or an unbound value if `authenticate` was called and generated an error.

See Also:

sendRedirect , authenticate

isValidTicketProvided

`bool $auth->isValidTicketProvided(void)`

Checks if the connecting client provided a valid, up to date application ticket. This method checks the following:

- Posession of a ticket
- Valid digital signature by the authenticator
- Renew time has not expired
- Client IP address matches that on ticket

Returns:

boolean indicating whether client submitted a valid application ticket via a cookie, or an unbound value if `authenticate` was called and generated an error.

isAnonymous

`bool $auth->isAnonymous(void)`

Checks if the user is accessing this application anonymously. This happens when a user declines to provide authentication information, or when the authenticator is configured to automatically return anonymous access to this application.

Returns:

boolean indicating whether the user is connected anonymously (has application ticket but no UserIID) , or an unbound value if `authenticate` was called and generated an error.

isAlternateUsernameProvided

`bool $auth->isAlternateUsernameProvided(void)`

Checks if the client's ticket contains an alternate username. Alternate usernames are provided on an application-specific basis by the authenticator to legacy applications which do not understand Intranet ID's, and require instead a different username be used by the user.

Returns:

boolean indicating whether the authenticator provided an alternate username for this application to use instead of UserIID , or an unbound value if `authenticate` was called and generated an error.

getAuthAlternateUsername

`string $auth->getAuthAlternateUsername(void)`

Returns the Authenticated Alternate Username if so included in the authentication ticket, or an empty string if none was included.

Returns:

the alternate username string provided by the authenticator, or an unbound value if `authenticate` was called and generated an error. If no alternate username was included in the ticket, returns an empty string.

See Also:

isAlternateUsernameProvided, authenticate

---

getAuthUserIID

`string $auth->getAuthUserIID(void)`

Returns the Intranet ID (IID) of the authenticated user, or an unbound value if `authenticate` was called and generated an error.

Returns:

the authenticated user IID string of the client or an unbound value if `authenticate` was called and generated an error. Returns an empty string if the user is anonymous.

---

getRedirectURL

`string $auth->getRedirectURL(void)`

If WAP requires a redirect, this call returns the URL to be redirected to or an unbound value if `authenticate` was called and generated an error.

Returns:

the URL which `sendRedirect` will redirect the client to, or an unbound value if `authenticate` was called and generated an error.

---

getRedirectType

`int $auth->getRedirectType(void)`

Returns REDIR_AUTH if WAP requires redirect to authenticator, REDIR_SELF if WAP requires redirect to self, and REDIR_NONE if WAP does not require a redirect.

Returns:

one of the defined integers REDIR_AUTH, REDIR_SELF or REDIR_NONE, reflecting what type of redirect is required by the WAP protocol. Returns an unbound value if `authenticate` was called and generated an error.

---

sendErrorPage

`void` *`$auth`*`->`**`sendErrorPage`**`(int` *`pageID`*`)`

Sends a fixed page out in accordance with WAP protocol.

Return:

This function does not return control to the calling process.

Parameters:

pageID - One of the defined integers BAD_SIG, BAD_CLIENT_IP or BAD_APP_IID

6.2 The TicketService Class

Signature

class **TicketService;**

Constructors

TicketService

```
$tktSvc=new TicketService(

        string ticketServiceURL,

        string ticketServicePublicKey    )
```

Creates a ticket service object to hold information defining a WAP authenticator ticket service

Parameters:

ticketServiceURL - the well-known URL of the ticket service

ticketServicePublicKey - the public key used to authenticate the authenticator

---

TicketService

```
$tktSvc = new TicketService(void)
```

Creates an uninitialized ticket service object

Methods

setURL

`void $tktSvc->setURL(string URL)`

Sets or replaces the URL representing the well-known URL of the ticket service.

Parameters:

URL - the well-known ticket service URL

setPublicKey

`void $tktSvc->setPublicKey(string publicKey)`

Sets or replaces the public key used to authenticate with this ticket service.

Parameters:

publicKey - the well-known ticket service public key string

getURL

`string $tktSvc->getURL(void)`

Return the well-known URL of the ticket service or an unbound value if not set.

getPublicKey

`string $tktSvc->getPublicKey(void)`

Return the well-known ticket service public-key, or an unbound value if not set.

Appendix C: DOC Intranet Cryptography Overview

Version: 1.0
Revised: 2000.11.12

1 Introduction

This document describes cryptographic methods used in the Department of Commerce intranet architecture. Cryptography is used to authenticate agents and to digitally sign documents. All cryptography used is asymetric, utilizing a public key and private key pair. Key management is beyond the scope of this document, but is assumed to occur in a directory service, in a PKI infrastructure, or by manual exchange.

Two components in the intranet architecture utilize cryptography: the Inter-Application Communication Protocol (IACP) and the Web-user Authentication Protocol (WAP). Additionally, various intranet HTTP connections utilize SSL-based encryption. SSL is not covered in this document.

The IACP uses digital signatures of nonce strings to authenticate parties participating in the protocol. These digital signatures are conveyed by way of HTTP headers. As such, they must be properly encoded to comply with HTTP header data transmission formats

WAP utilizes digital signatures of application tickets and session ticket. Both tickets are transmitted in plaintext as HTTP headers. The digital signatures are transmitted as HTTP headers, and must be properly encoded in a compliant HTTP header data transmission format.

This document covers two aspects: digital signatures used in WAP and IACP, and the encoding technique used to transmit encrypted data via HTTP headers.

2 Digital Signatures

A digital signature is a binary string of data created using the signer's private key. As such, it cannot be forged by anyone who does not possess the signer's private key.

For the purposes of the intranet architecture, all digital signatures are constructed using techniques outlined in [FIPS186-2], Digital Signature Standard. The Digital Signature Standard mandates SHA-1 for the message digest portion of the digital signature, but allows a choice of algorithms for the digital signing step. All digital signatures covered by this document must use the RSA digital signing algorithm.

3 Encoding for HTTP Transmission

Digital signatures as generated by the techniques outlined in section 2 are binary data, and as such are not suitable for direct inclusion in a HTTP header. The BASE64 encoding scheme (see [RFC1521], Section 5.2) provides a method of encoding binary data into a string consisting exclusively of the characters A-Z, a-z, 0-9, +, / and =.

There are two variants of this algorithm in common use. The first, which strictly abides by [RFC1521], limits the length of an encoded string to a maximum of 76 characters. This is suitable for email headers, but not appropriate for many other general-purpose encodings. The second variant removes this limitation, and encodes arbitrarily long strings. For use in HTTP headers within the Department of Commerce's intranet, the second technique will be utilized.

Further, all encoded header values are to be enclosed in double quotes.

As an example, consider the case where the following digital signature has been created.

Digital Signature: `akZ&"9zl^@~'"!73`

Applying the BASE64 encoding scheme yields the following.

Base64 Encoding: `YWtaJiI5emxeQH4nIiE3Mw==`

Finally, to transmit this signature in a header named APP_TICKET, the following header line is included.

APP_TICKET="`YWtaJiI5emxeQH4nIiE3Mw=="`

4 References

[FIPS186-2] FIPS Pub 186-2: Digital Signature Standard, available http://csrc.nist.gov/fips/fips186-2.pdf

[RFC1521] MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, available http://www.faqs.org/rfcs/rfc1521.html

Appendix D: Directory Schema for the DOC Intranet Directory Service

Version 1.0

Revised: 2000.11.12

1 Introduction

The directory service stores information about users and online applications in the DOC intranet, as well as the complete organizational structure and other groups within DOC. This document describes the schema used within the directory service.

2 Terminology

attribute

> A single unit of information in an LDAP entry with an associated syntax. Analogous to a field in a database record.

API

> Application Programming Interface. A set of routines provided for a developer to access programmatic functionality that has already been developed.

ces

> Case Exact String. The syntax method for LDAP attributes that indicates that values for this attribute are case sensitive.

cis

> Case Ignore String. The syntax method for LDAP attributes that indicates that values for this attribute are not case sensitive.

cn

> commonName. A common LDAP attribute. The textual or spoken proper name of an entry.

dn

> Distinguished Name. The fully qualified name of an LDAP entry. It is unique within the directory, and comprised of the RDNs of the entry from root.

DOC

Department of Commerce.

IANA

Internet Assigned Numbers Authority. The central coordinator for the assignment of unique parameter values for Internet protocols.

IACP

Inter-Application Protocol. The Protocol used within the DOC intranet to allow online applications to share information.

IETF

Internet Engineering Task Force. The international community of network designers, operators, vendors, and researchers concerned with the evolution of Internet architecture. Working groups within the IETF administer Requests For Comment (RFCs) that define much of what has become standard in Internet architecture.

IID

Intranet ID. The identifier for any entry in the DOC intranet directory service. The

IID is unique in the entire directory service. For instance, an application cannot have the same IID as a person. It is a string composed of numbers and upper and lower case letters, and is case-sensitive.

LDAP

Lightweight Directory Access Protocol. Version 3 is defined in RFC2251. While the protocol itself does not describe implementation details for a directory service, in common usage the term is used to describe a directory server that supports the LDAP protocol.

objectclass

An entry in an LDAP directory. It is comprised of a collection of required and optional attributes. Although it is analogous to a record in a database, attributes can be multivalued- i.e. a single entry may contain more than one value for a given attribute.

OID

Object Identifier. Unique identifier used to identify LDAP objectclasses, attributes, and syntax descriptions recognized by the Internet Engineering Task Force (IETF). Assigned by Internet Assigned Numbers Authority (IANA).

ou

> Organizational Unit. A common LDAP attribute that has two conventional uses: to contain pieces of a formal organization, such as Departments or Offices; or to differentiate between categories within the directory itself, such as People and Printers. For the schema described here, it is always used in the latter sense.

RDN

> Relative Distinguished Name. The identifier which is unique to a given entry at its level in the hierarchy. It is always the leftmost portion of an entry's dn.

uid

> The LDAP abbreviation for userid. This is an optional attribute for **docAppUser** entries which will be used when the **docApp** is a legacy application.

3 Overview

3.1 Directory Service Access

The directory service can be accessed for user and application information directly by a DOC online application, but will typically be accessed via the Directory API. Group information, while stored in the directory service, is accessed only by the Group Service. The Group Service is accessed via Inter-Application Communication Protocol (IACP).

Figure 1: Directory Subsystem Diagram, summarizes access to the Directory Service.

Access controls are described for each objectclass. The privileges described in this document pertain to the normal users of the system. Other privileges may be granted to a DOC intranet "superusers" group, a directory service maintenance group, and other resource management groups as DOC policy dictates, but are not included here.

For a general discussion of how LDAP access control is specified, see the OpenLDAP 2.0 Administrator's Guide:

> http://www.openldap.org/doc/admin/slapdconfig.html#Access Control

3.2 Directory Service Organization

There are three organizational units (ou's) from root: ***apps***, ***people***, and ***groups***. Every application, person, and group has a unique identifier attribute, **docIid**, where IID denotes intranet ID.

Each application has a **docApp** entry that specifies public information about each online application.

Each person has one **docPerson** entry, and zero or more **docAppUser** entries. The **docPerson** entry contains conventional directory information for the person, such as addresses and telephone numbers. Each **docAppUser** entry contains application-specific information pertaining to the person.

The ***groups*** ou contains contains a further organizational level. This level will specify the type of group. Initial entries at this level will include ***DOC Org***, ***External Org***, ***Admins***, and ***User Defined***. Under this level live **docGroup** entries.

Figure 2: Directory Schema Diagram, summarizes the schema organization.

3.3 Common Attributes

The attributes shown below are used by most objectclasses defined in this document.

Attribute Name	Syntax	OID
ObjectClass	cis, 1-many	2.5.4.0
Specifies the object classes of the object. Must include the object.		
DocIid	ces, 1	(local)
Specifies the Intranet ID (IID) of the entry. This identifier is specific to the DOC intranet. Legal characters are numerals and upper and lower case characters. The IID is used by the intranet infrastructure and associated programs: it is never entered or seen by a user of the system. Note that the syntax is case exact string, so an IID of 1wR4 is different from 1WR4.		
Cn	cis, 1-many	2.5.4.3
commonName. Identifies the name of an object in the directory. When the object corresponds to a person, the cn it is will be the person's full name.		

4 Application Entries

4.1 docApp Entries

4.1.1 The docApp Objectclass

Under the ***apps*** ou reside **docApp** entries. **docApp** entries contain public information about each online application. The RDN for each **docApp** entry is **docIid**.

Note that a particular application may have several instances running within the DOC intranet. For example, several offices may each use their own installation of a given application. In this case, each instance of the application will have a separate **docApp** entry. As far as the directory is concerned, they are unrelated applications. The **docCnAbbrev** attribute must be unique among **docApp**s.

```
    objectClass docApp

        requires

            objectClass,

            docIid,

            cn,

            docCnAbbrev,

            docEntryUrl,

            docIacpUrl,

            docRequireSsl

        allows

            description,

            docReturnUrlDomain,

            docReturnUrlPath,

            docAdmin,

            docGroupAuthorized
```

4.1.2 docApp Attributes

Attribute Name	Syntax	OID
DocCnAbbrev	cis, 1	(local)
An abbreviated version of the commonName (cn) attribute. This must be unique among **docApp**s. It should be easily to remember and refer to by humans, as it will be a useful way to refer to an instance of a particular application by developers and frequent users of an application.		
DocEntryUrl	ces, 1	(local)
The URL by which a user enters the application.		
DocIacpUrl	ces, 1	(local)

The URL used for Inter-Application Protocol requests.		
DocRequireSsl	boolean, 1	(local)
Whether or not authenticated users with an application ticket must access the application via an SSL connection.		
Description	cis	2.5.4.13
Description of the application.		
DocReturnUrlDomain	cis, 1	(local)
The domain field of a cookie used by the application.		
DocReturnUrlPath	ces, 1	(local)
The path field of a cookie used by the application.		
DocAdmin	ces	(local)
The IID of the **docGroup** or **docPerson** that manages the application. If it is a group, it will normally be in the ***Admins*** group ou (see section 6.1).		
DocGroupAuthorized	ces	(local)
The IID of a group authorized to use the application.		

4.1.3 docApp Access Control

A group of **docApp** entries are granted read and lesser privileges (auth, compare, and search) to the following attributes: **docIid**, **cn**, **docEntryUrl**, **docIacpUrl**, **docRequireSsl**, **docCnAbbrev**, **description**, **docReturnUrlDomain**, and **docReturnUrlPath**. This group will include the Portal Server. As new applications are developed that require read access to **docPerson** entries, they must be added to that group.

The WAP server and Group Server are granted read and lesser privileges to all of the above attributes, as well as **docGroupAuthorized**

The Directory Management **docApp** has full privileges. Users of the DOC intranet manipulate directory entries solely through this application. While it has full privileges to directory contents, it will govern an individual's ability to view, add, or modify directory contents based on its own internal configuration.

4.1.4 Example docApp Entry

```
dn: docIid=r4gF6, ou=apps, o=doc.gov

objectClass: top

objectClass: docApp

docIid: r4gF6

cn: DOC Sample Online App for OHRM

docCnAbbrev: SampleApp-OHRM

description: The DOC sample application used by OHRM

docEntryUrl: http://host2.doc.gov/sampleApp/ohrm/

docIacpUrl: http://host2.doc.gov/sampleApp/ohrm/iacp

docRequireSsl: TRUE

docReturnUrlDomain: host2.doc.gov

docReturnUrlPath: /sampleApp/ohrm

docAdmin: r4h02

docGroupAuthorized: r4h01

docGroupAuthorized: r4h35
```

5 People Entries

5.1 docPerson Entries

5.1.1 The docPerson Objectclass

The highest objectClass under ***people*** is **docPerson**. The RDN for a **docPerson** is **docIid**.

For a person with multiple addresses, a postalAddress attribute will contain each non-primary address, while the primary address is divided into the other address part attributes to facilitate location-based searching.

```
objectClass docPerson

    requires

        objectClass,

        docIid,

        cn
```

```
allows
    sn,
    givenName,
    generationQualifier,
    displayName,
    docShortDisplayName,
    mail,
    telephoneNumber,
    facsimileTelephoneNumber,
    mobile,
    pager,
    docOtherPhoneNumber,
    buildingName,
    roomNumber,
    postalAddress,
    streetAddress,
    physicalDeliveryOfficeName,
    st,
    postalCode
```

5.1.2 docPerson Attributes

Attribute Name	Syntax	OID
Sn	cis	2.5.4.4
surName. The person's surname, also referred to as last name or family name.		
GivenName	cis	2.5.4.42
Identifies the entry's given name, usually a person's first name.		

GenerationQualifier	cis	2.5.4.44
Contains the generation Qualifier part of the name, typically appearing in the suffix, for example: `generationQualifier: III`		
DisplayName	cis	2.16.840.1.113730.3.1.241
Preferred name of a person to be used when displaying entries. Especially useful in displaying a preferred name for an entry within a one-line summary list. Since other attribute types, such as cn, are multivalued, they can not be used to display a preferred name. For example: `displayName: Rob Wilson`		
docShortDisplayName	cis	(local)
A short version of the display name. For example: `docShortDisplayName: Rob`		
Mail	cis	0.9.2342.19200300.100.1.3
The person's primary email address.		
TelephoneNumber	tel	2.5.4.20
The person's telephone number.		
Fax	tel	2.5.4.23
facsimileTelephoneNumber. The fax number at which the person can be reached.		
Mobile	tel	0.9.2342.19200300.100.1.41
mobileTelephoneNumber. The person's mobile or cellular phone number.		
Pager	tel	0.9.2342.19200300.100.1.42
pagerTelephoneNumber. The person's pager telephone number.		
docOtherPhoneNumber	cis	2.5.4.20

Another telephone numbers associated with the person followed by a short description of the significance of the number. For example: `docOtherPhoneNumber: 202-555-1234 telecommuting`		
BuildingName	cis	0.9.2342.19200300.100.1.48
The building name associated with the person.		
RoomNumber	cis	0.9.2342.19200300.100.1.6
The room number associated with the person.		
PostalAddress	cis	2.5.4.16
Identifies the entry's mailing address. This field is intended to include multiple lines. When represented in LDIF format, each line should be separated by a dollar sign ($). To represent an actual dollar sign ($) or backslash (\) within this text, use the escaped hex values \24 and \5c respectively.		
Street	cis	2.5.4.9
streetAddress. The person’s street number and name, for example: `street: 6789 Slippery Slope Rd NW`		
physicalDeliveryOfficeName	cis	2.5.4.19
Identifies the name of the city or village where a physical delivery office is located. For example: `physicalDeliveryOfficeName: Blacksburg`		
St	cis	2.5.4.8
stateOrProvinceName. Identifies the state or province in which the entry resides.		
PostalCode	cis	2.5.4.17
The person’s zip code (US) or postal code.		

5.1.3 docPerson Access Control

A group of **docApp** entries are granted read and lesser privileges to all attributes. This group will include the Group Server. As new applications are developed that require read access to **docPerson** entries, they must be added to that group.

The Directory Management **docApp** has full privileges. Privileges can be extended selectively to users of the Directory Management application (such as the ability of an employee to change their preferred name) based on rules internal to it.

5.1.4 Example docPerson Entry

```
dn: docIid=r4gE2, ou=people, o=doc.gov

objectClass: top

objectClass: docPerson

docIid: r4gE2

cn: Roberto J. Wilson

sn: Wilson

givenName: Roberto

displayName: Rob Wilson

docShortDisplayName: Rob

mail: wilsonrj@doc.gov

telephoneNumber: 202-555-4567

facsimileTelephoneNumber: 202-555-4568

mobile: 202-555-4569

pager: 202-555-4570

docOtherPhoneNumber: 202-555-1234 telecommuting

buildingName: Building XII

roomNumber: 3700

postalAddress: PO Box 27553$Blacksburg, VA 24063

streetAddress: 6789 Slippery Slope Rd NW

physicalDeliveryOfficeName: Blacksburg

st: VA

postalCode: 24060
```

5.2 docAppUser Entries

5.2.1 The docAppUser Objectclass

A **docAppUser** object specifies attributes for a **docPerson** associated with a particular **docApp**. The **docAppUser** class or any class extending it would be placed under its corresponding **docPerson** entry in the schema hierarchy.

The dn for this object will be composed of the **docIid** of the application, the **docIid** of the person, the ***people*** ou, and the ***doc.gov*** o. The dn's of the pertinent **docPerson** and **docApp** are required attributes of the entry.

The **docAppUser** objectclass itself contains only one optional attribute, **uid**. This is used when the user's application userID (logon id) is different from their intranet userID. If an application wishes to use the directory service to store other parameters for each user, it is necessary to extend the **docAppUser** objectClass.

```
objectClass docAppUser

    requires

        objectClass,

        docPerson,

        docApp

    allows

        uid
```

5.2.2 docAppUser Attributes

Attribute Name	Syntax	OID
DocPerson	ces	(local)
The person to whom the entry applies. Specified by dn.		
DocApp	ces	(local)
The application to which the entry applies. Specified by dn.		
Uid	cis	0.9.2342.19200300.100.1.1
userID. The person's logon id for the given application.		

5.2.3 docAppUser Access Control

Full privileges are granted to the **docApp** to which the entry refers for the **uid** attribute.

The Directory Management **docApp** has full privileges. Privileges can be extended selectively to users of the Directory Management application based on rules internal to it.

5.2.4 Example docAppUser Entry

```
dn: docIid=r4gF6,docIid=r4gE2,ou=people,o=doc.gov

objectClass: top

objectClass: docAppUser

docPerson: docIid=r4gE2, ou=people, o=doc.gov

docApp: docIid=r4gF6, ou=apps, o=doc.gov

uid: wilsonr4
```

6 Groups Entries

6.1 Types of Groups

There is a single organizational unit (ou) level beneath the ***group's*** ou that specifies the type of group. Initial entries at this level will include ***DOC Org***, ***External Org***, ***Admins***, and ***User Defined***. As the DOC intranet is developed and once deployed, entries may be added at this level to allow new types of groups.

Entries under the second level ou's are of objectclass **docGroup**.

6.2 Organizational Groups

Organizational groups for most entities needing directory services are hierarchical, so a great deal of effort was expended in the early years of directory technology to make the hierarchy of the groups and subgroups the guideline for the directory hierarchy.

Experience revealed that large complex entities undergo far too many structural changes to continue to reorganize them based on the group structure.

For the DOC intranet, the directory server is not being used just for typical directory information lookup. It is also providing infrastructure information for the intranet to support application needs such as authentication and authorization. For group information, it must answer queries that are highly inefficient for both traditional directory structures and relational databases.

The Group Server (see *Figure 1: Directory Subsystem Diagram*) will help service these queries. It will perform a large number of queries to the directory service upon initialization or reload to obtain the organizational group entries. Then it will cache the information in a data structure that maximizes efficiency for the group queries.

The **docGroup**s that live within a certain ou will all be directly under that ou, and will contain an attribute that specifies the parent group. For the highest group in any hierarchy, the parent attribute will contain the keyword ***null***. This will allow an easy means for the Group Server to construct its data structure and allow easy modification of the group structure for system administrators.

6.3 docGroup Entries

6.3.1 The docGroup Objectclass

docGroup entries contain information about each group, the group's parent group, the group's managing group or person, and a list of group members. The RDN for each **docGroup** entry is **docIid**.

For a top-level group, the **docParentGroup** attribute contains the keyword ***null***.

The **docAdmin** can contain the IID of an individual or another group.

```
objectClass docGroup

    requires

        objectClass,

        ou,

        docIid,

        cn,

        docCnAbbrev,

        docParentGroup,

        docAdmin

    allows

        description,

        docInfoUrl,

        docMember
```

6.3.2 docGroup Attributes

Attribute Name	Syntax	OID
Ou	cis	2.5.4.11
organizationalUnitName. For a **docGroup** entry, this will contain the ou under the *groups* ou to which the group belongs.		
DocCnAbbrev	cis, 1	(local)
An abbreviated version of the commonName (cn) attribute. This must be unique among **docGroup**s. It should be easily to remember and refer to by humans, as it will be a useful way to refer to a group by developers and group members.		
DocParentGroup	cis, 1	(local)
The dn of the parent group of this group. If the group is at the top of its hierarchy, then it contains the keyword ***null***.		
DocGroupAdmin	ces	(local)
The IID of the individual or group that manages the group. It is legitimate for this value to be self-referential- for all members of the group to be able manage the group.		
Description	cis	2.5.4.13
Description of the group.		
DocInfoUrl	ces	(local)
URL associated with the group.		
DocMember	ces	(local)
One entry for each member of the group. The IID can refer to an individual or a group.		

6.3.3 docGroup Access Control

Full privileges are granted to the Group Server. Since even the Directory Management application will use the Group Server for group information reading and changes, there is no need to grant privilege to it.

6.3.4 Example docGroup Entry

```
dn: docIid=r4h01, ou=DOC Org, ou=groups, o=doc.gov

objectClass: top

objectClass: docGroup
```

```
docIid: r4h01
cn: Office of Group Examples
docParentGroup: r4h00
docAdmin: r4gYi
docCnAbbrev: OOGE
description: An example group for the docOrgGroup objectclass
docInfoUrl: http://host12.doc.gov/exampleGroups/OOGE/
docMember: u483i
docMember: K8Rre
docMember: PHk2m
docMember: E3w22
docMember: v7Jes
```

7 References

[RFC2251] Lightweight Directory Access Protocol (v3)

http://www.faqs.org/rfcs/rfc2251.html

Internet Engineering Task Force

http://www.ietf.org/

Internet Assigned Numbers Authority

http://www.iana.org/

Netscape Universal Schema Reference

http://developer.netscape.com/docs/manuals/directory/schema2/41/contents.htm

OpenLDAP 2.0 Administrator’s Guide

http://www.openldap.org/doc/admin/slapdconfig.html

Figure 1: Directory Subsystem

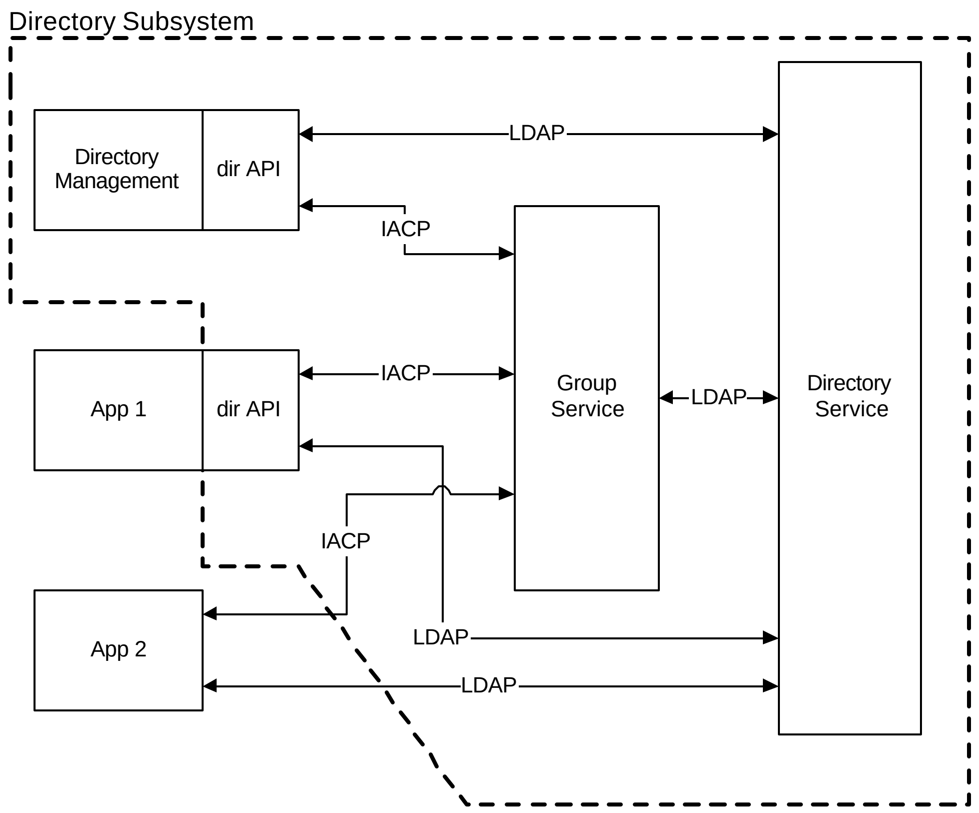
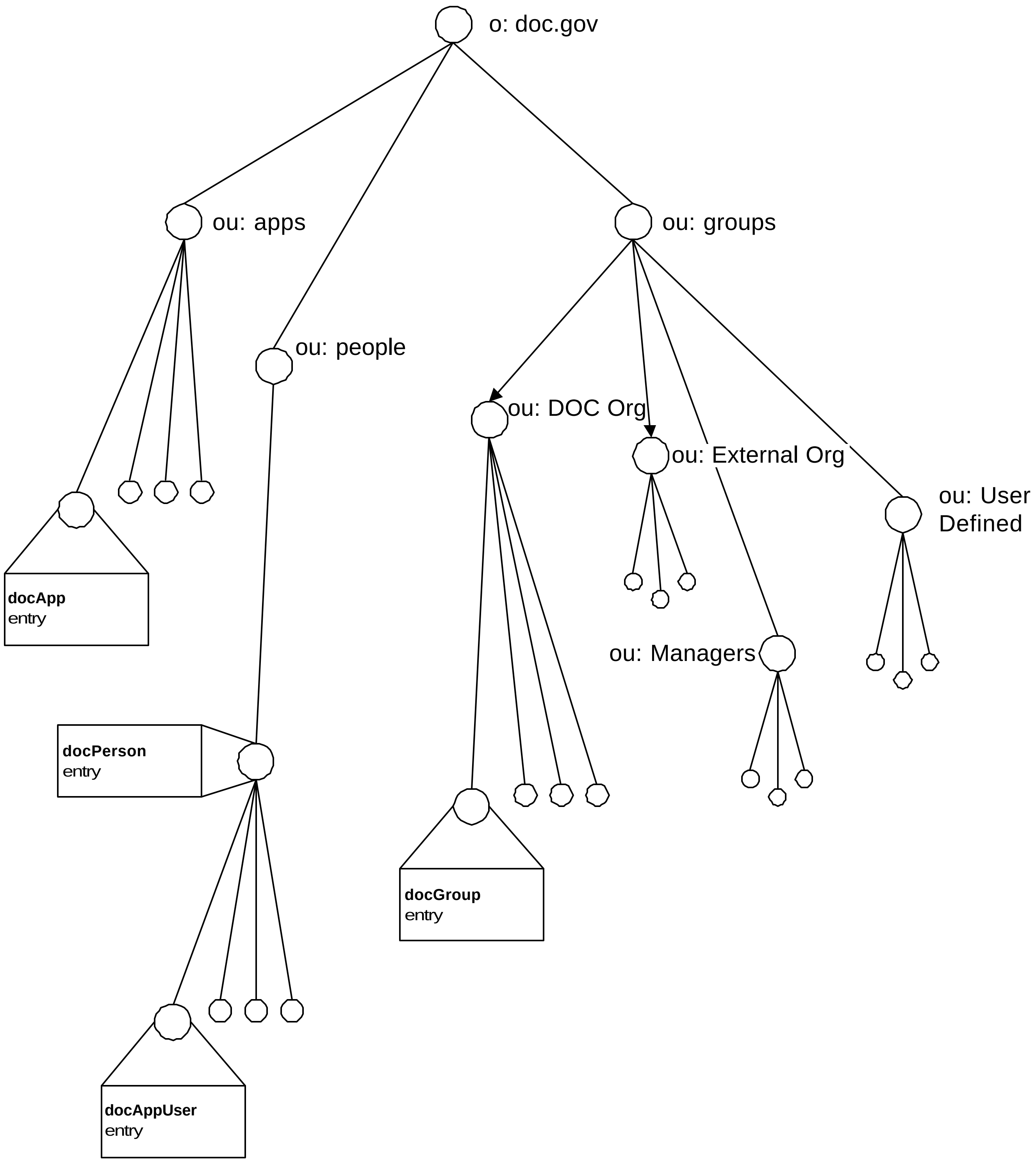


Figure 2: Directory Schema



Appendix E: Directory API for the DOC Intranet Directory Service

Version 1.0

Revised: 2000.11.12

1 Introduction

Directory service and Group Service information needs to be accessed by many different applications within the DOC intranet. The purpose of the API is to provide easy access to that information. This document describes the API for Java, Perl, and PHP.

2 Terminology

attribute

> A single unit of information in an LDAP entry with an associated syntax. Analogous to a field in a database record.

API

> Application Programming Interface. A set of routines provided for a developer to access programmatic functionality that has already been developed.

dn

> Distinguished Name. The fully qualified name of an LDAP entry. It is unique within the directory, and comprised of the RDNs of the entry from root.

DOC

> Department of Commerce.

IID

> Intranet ID. The identifier for any entry in the DOC intranet directory service. The
>
> IID is unique in the entire directory service. For instance, an application cannot have the same IID as a person. It is a string composed of numbers and upper and lower case letters, and is case-sensitive.

LDAP

> Lightweight Directory Access Protocol. Version 3 is defined in RFC2251. While the protocol itself does not describe implementation details for a directory service, in

common usage the term is used to describe a directory server that supports the LDAP protocol.

RDN

Relative Distinguished Name. The identifier that is unique to a given entry at its level in the hierarchy. It is always the leftmost portion of an entry's dn.

3 Overview

3.1 Using the Directory API

The Directory API provides applications an easy means of accessing DOC directory service and Group Service information. Rather than making calls to the LDAP server and group server directly, the application uses API calls that deal with networking, protocol, and data handling issues. The programmer need only concentrate on the application they are trying to build.

A diagram showing the Directory API's place in the directory portion of the DOC intranet infrastructure is included in *Figure 1: Directory Subsystem Diagram*.

3.2 The *docDirectory.conf* File

A file named *docDirectory.conf* is associated with each of the libraries that implement the Directory API. The file contains the hostname and port of the directory service, the hostname and port of the Group Service, and the application IID and password for binding. The following is a sample *docDirectory.conf*.

```
#########################################################

# DocDirectory.conf

#

# Configuration file for the DOC Directory API library

#########################################################



# ldapHost=<hostname>

# ldapPort=<port>



ldapHost=myhost.mydomain.gov

ldapPort=636
```

```
# groupServerHost=<hostname>

# groupServerPort=<port>


groupServerHost=myhost.mydomain.gov

groupServerPort=7000


# appIid=<iid>

# appPassword=<password>


appIid=zU8oS

appPassword=myapppassword
```

4 The Directory API for Java

4.1 The DocDirectory Class

To use the Directory API in Java, one obtains a DocDirectory object and calls its methods.

The DocDirectory object is obtained by calling the static `getInstance` method on the DocDirectory class. The `getInstance` method takes a single parameter, the name of the *docDirectory.conf* file. The contents of the file are discussed in detail in the next section.

Once the DocDirectory object is obtained, a variety of methods are available to obtain information from the directory service and Group Service.

4.2 Obtaining the DocDirectory Class

```
public static DocDirectory getInstance(String confFileName)
    throws BadConfigParametersException FIXME
```

Parameters:

`ConfFileName` - The name of the *docDirectory.conf* file including path if necessary.

Returns:

A DocDirectory object initialized with the contents of the supplied conf file.

Throws:

`BadConfigParametersException` – The contents of the *docDirectory.conf* file did not enable the DocDirectory object to establish a connection to the directory service.

4.3 Retrieving App Information in Java

`public boolean` **`isApp`**`(String iid)`

Parameters:

`iid` – The IID of a potential application.

Returns:

True if the IID corresponds to a **docApp** entry in the directory.

`public Properties` **`getAppAttributes`**`(String appIid)`

`throws NoSuchAppException`

Parameters:

`appIid` – The IID of the application.

Returns:

A Properties object containing all of the application's attributes for which the requesting entity has read privilege.

Throws:

`NoSuchAppException` – The IID supplied either does not exist, or does not correspond to a **docApp** entry.

4.4 Retrieving Person Information in Java

`public boolean` **`isPerson`**`(String iid)`

Parameters:

`iid` – The IID of a potential person.

Returns:

True if the IID corresponds to a **docPerson** entry in the directory.

```
public String getIidForUsername(String username)
      throws NoSuchPersonException
```

Parameters:

`username` – The intranet uid of the person.

Returns:

The IID of the person corresponding to the username.

Throws:

`NoSuchPersonException` – The username supplied does not exist.

```
public Properties getPersonAttributes(String userIid)
      throws NoSuchPersonException
```

Parameters:

`userIid` – The IID of the person.

Returns:

A Properties object containing all of the person's attributes for which the requesting entity has read privilege.

Throws:

`NoSuchPersonException` – The IID supplied either does not exist, or does not correspond to a **docPerson** entry.

```
public Vector getPeopleAttributes(String[] userIids,
                                  String[] attributes)
      throws NoSuchAttributeException, InsufficientPrivilegeException
public Vector getPeopleAttributes(Vector userIids,
                                  Vector attributes)
      throws NoSuchAttributeException, InsufficientPrivilegeException
```

Parameters:

`userIid` – An array or Vector of Strings that identify **docPerson** entries.

`attributes` – An array or Vector of Strings containing the attributes that are to be returned. It is not necessary to include `docIid` in this list, as it is always returned.

Returns:

A Vector containing Properties objects. Each Properties object contains one **docPerson**’s attributes for which the requesting entity has read privilege. If an object that is not a String is in the Vector or a String that does not correspond to a **docPerson** is encountered, it is ignored: no Exception is thrown.

Throws:

`NoSuchPersonException` – The IID supplied either does not exist, or does not correspond to a **docPerson** entry.

`InsufficientPrivilegeException` – The binding entity does not have the necessary privilege to read one or more attributes requested.

```
public boolean isPersonInGroup(String userIid,
                               String groupIid)
     throws NoSuchPersonException, NoSuchGroupException
public boolean isPersonInGroup(String userIid,
                               String groupIid,
                               boolean recurse)
     throws NoSuchPersonException, NoSuchGroupException
```

Parameters:

`userIid` – The IID of the person.

`groupIid` – The IID of the group.

`recurse` - True causes the method to examine all subgroups for membership. False causes the method to consider only direct membership in the group. If the parameter is not present, defaults to true.

Returns:

True if the user is a member of the group, false otherwise.

Throws:

`NoSuchPersonException` – The IID supplied either does not exist, or does not correspond to a **docPerson** entry.

`NoSuchGroupException` – The IID supplied either does not exist, or does not correspond to a **docGroup** entry.

```
public Vector getGroupsForPerson(String userIid)

      throws NoSuchPersonException

public Vector getGroupsForPerson(String userIid, boolean recurse)

      throws NoSuchPersonException
```

Parameters:

`usersIid` – The IID of the person for whom group information is being requested.

`recurse` – True causes the method to return all supergroups. False causes the method to return only groups in which the person has direct membership. If the parameter is not present, defaults to true.

Returns:

A Vector containing Strings that correspond to the IIDs of groups in which the user is a member.

Throws:

`NoSuchPersonException` – The IID supplied either does not exist, or does not correspond to a **docPerson** entry.

```
public Vector getSubGroupsForUser(String userIid,

                                      String superGroupIid)

      throws NoSuchPersonException, NoSuchGroupException
```

Parameters:

`userIid` – The IID of the person.

`superGroupIid` – The IID of the group.

Returns:

A Vector containing Strings that correspond to the IIDs of all subgroups that are hierarchically under the superGroup in which the user is a member.

Throws:

`NoSuchPersonException` – The IID supplied either does not exist, or does not correspond to a **docPerson** entry.

`NoSuchGroupException` – The IID supplied for superGroup either does not exist, or does not correspond to a **docGroup** entry.

4.5 AppUser Information

4.5.1 Retrieving AppUser Information in Java

```
public Properties getAppUserAttributes(String userIid, String appIid)

     throws NoSuchPersonException, NoSuchAppException
```

Parameters:

`userIid` – The IID of the person.

`appIid` – The IID of the application.

Returns:

A Properties object containing all of the appUser's attributes for which the requesting entity has read privilege.

Throws:

`NoSuchPersonException` – The IID supplied either does not exist, or does not correspond to a **docPerson** entry.

`NoSuchAppException` – The IID supplied for app either does not exist, or does not correspond to a **docApp** entry.

4.5.2 Setting AppUser Information in Java

```
public void setAppUserAttributes(String userIid,

                                 String appIid,

                                 Properties changedAttributes)

     throws NoSuchPersonException, NoSuchAppException,
            NoSuchAttributeException
```

Parameters:

`userIid` – The IID of the person.

`appIid` – The IID of the application.

`changedAttributes` – A Properties object containing the new values of the attributes that are to be changed.

Throws:

`NoSuchPersonException` – The IID supplied either does not exist, or does not correspond to a **docPerson** entry.

`NoSuchAppException` – The IID supplied for app either does not exist, or does not correspond to a **docApp** entry.

`NoSuchAttributeException` – One or more attributes in the changedAttributes Properties does not exist in the **docAppUser** entry.

`InsufficientPrivilegeException` – The binding entity does not have the necessary privilege to change one or more attributes it is attempting to change.

4.6 Retrieving Group Information in Java

`public boolean` **`isGroup`**`(String iid)`

Parameters:

`iid` – The IID of a potential group.

Returns:

True if the IID corresponds to a **docGroup** entry in the directory.

`public Properties` **`getGroupAttributes`**`(String groupIid)`

`throws NoSuchGroupException`

Parameters:

`groupIid` – The IID of the group.

Returns:

A Properties object containing all of the group's attributes for which the requesting entity has read privilege.

Throws:

`NoSuchGroupException` – The IID supplied either does not exist, or does not correspond to a **docGroup** entry.

```
public int getGroupMemberCount(String groupIid)
     throws NoSuchGroupException
public int getGroupMemberCount(String groupIid, boolean recurse)
     throws NoSuchGroupException
```

Parameters:

`groupIid` – The IID of the group.

`recurse` – True causes the method to count members of the group and subgroups. False causes only immediate members to be counted. If `recurse` is not specified, it is true by default.

Returns:

The number of members in the group.

Throws:

`NoSuchGroupException` – The IID supplied either does not exist, or does not correspond to a **docGroup** entry.

```
public Vector getGroupMembers(String groupIid, boolean recurse)
     throws NoSuchGroupException
```

Parameters:

`groupIid` – The IID of the group.

`recurse` – True causes the method to return members of the group and all subgroups. False causes only immediate **docPerson** members to be returned. If `recurse` is not specified, it is true by default.

Returns:

A Vector containing Strings which correspond to the IIDs of the members of the group. A member corresponds to a **docPerson**, not a subgroup of the group.

Throws:

`NoSuchGroupException` – The IID supplied either does not exist, or does not correspond to a **docGroup** entry.

```
public boolean isGroupWithinGroup(String groupIid,
                                  String superGroupIid)
   throws NoSuchGroupException
public boolean isGroupWithinGroup(String groupIid,
                                  String superGroupIid,
                                  boolean recurse)
     throws NoSuchGroupException
```

Parameters:

`groupIid` – The IID of the subGroup.

`superGroupIid` – The IID of the superGroup.

`recurse` - True causes the method to examine the group and all subgroups for membership. False causes the method to consider only immediate member groups. If `recurse` is not specified, it is true by default.

Returns:

True if the group is hierarchically contained by the superGroup, false otherwise.

Throws:

`NoSuchGroupException` – The IID supplied either does not exist, or does not correspond to a **docGroup** entry.

```
public String getParentGroup(String groupIid)
     throws NoSuchGroupException
```

Parameters:

`groupIid` – The IID of the group.

Returns:

The IID of the immediate parent of the group. If the group is the highest in its hierarchy, returns an empty String.

Throws:

`NoSuchGroupException` – The IID supplied either does not exist, or does not correspond to a **docGroup** entry.

```
public Vector getSubGroups(String groupIid)

      throws NoSuchGroupException

public Vector getSubGroups(String groupIid, boolean recurse)

      throws NoSuchGroupException
```

Parameters:

`groupIid` – The IID of the group.

`recurse` - True causes the method to return all member groups and subgroups. False causes only immediate member groups to be returned. If `recurse` is not specified, it is true by default.

Returns:

A Vector containing Strings corresponding to the IIDs of the immediate subGroups of the group. If the group is the lowest in its hierarchy, returns an empty Vector.

Throws:

`NoSuchGroupException` – The IID supplied either does not exist, or does not correspond to a **docGroup** entry.

5 The Directory API for Perl

5.1 The DocDirectory Object

The Perl Directory API library resides in a module named DocIntranet, which is contained in the file *DocIntranet.pm*. To use the Directory API in Perl, one creates a DocDirectory object and calls methods on it.

The DocDirectory object's handle is obtained by calling the `new` method of the DocDirectory class. The `new` method takes the name of the *docDirectory.conf* file as its only parameter.

Once the DocDirectory handle is obtained, a variety of methods are available to obtain information from the directory service and Group Service.

5.2 Obtaining a DocDirectory Handle

new

```
$docdir = new DocDirectory docdirectoryconffilename
```

Constructor. Creates a new DocDirectory handle initialized with the parameters of the *docDirectory.conf* file. The *docdirectoryconffilename* parameter may include the file's path if necessary.

5.3 Retrieving App Information in Perl

is_app

```
$docdir->is_app(iid)
```

Returns true if the iid corresponds to a **docApp** entry in the directory.

get_app_attrs

```
$docdir->get_app_attrs(appiid)
```

Returns a hash containing all of the application's attributes for which the requesting entity has read privilege.

5.4 Retrieving Person Information inPerl

is_person

```
$docdir->is_person(iid)
```

Returns true if the iid corresponds to a **docPerson** entry in the directory.

get_iid_for_uname

`$docdir->get_iid_for_uname(username)`

Returns a scalar containing the IID of the user.

get_person_attrs

`$docdir->get_person_attrs(useriid)`

Returns a hash containing all of the user's attributes for which the requesting entity has read privilege.

get_people_attrs

`$docdir->get_people_attrs(useriids, attributes)`

Returns an array filled with hashes. Each hash contains one **docPerson**'s attributes for which the requesting entity has read privilege. If any item in the `useriids` array does not correspond to a **docPerson** entry, it is ignored.

Takes the following arguments:

useriids

An array containing the IIDs that identify **docPerson** entries.

attributes

An array containing the attributes that are to be returned. It is not necessary to include `docIid` in this list, as it is always returned.

is_person_in_group

`$docdir->is_person_in_group(useriid, groupiid[, recurse])`

Returns true if the person is a member of the group, false otherwise. If the recurse parameter is true, the method examines all subgroups for membership. False causes the method to consider only direct membership in the group. If the parameter is not present, defaults to true.

get_groups_for_person

`$docdir->get_groups_for_person(useriid[, recurse])`

Returns an array containing the IIDs of groups in which the user is a member. The keys of the array are numeric and arbitrarily assigned. If `recurse` is true, the method returns all supergroups. False causes the method to return only groups in which the person has direct membership. If the parameter is not present, defaults to true.

get_subgroups_for_person

*$docdir->***get_subgroups_for_person***(useriid, supergroupiid)*

Returns an array containing the IIDs of all subgroups that are hierarchically under the supergroup in which the user is a member.

5.5 AppUser Information

5.5.1 Retrieving AppUser Information in Perl

get_appuser_attrs

*$docdir->***get_appuser_attrs***(useriid, supergroupiid)*

Returns a hash containing all of the appUser's attributes for which the requesting entity has read privilege.

5.5.2 Setting AppUser Information in Perl

set_appuser_attrs

*$docdir->***set_appuser_attrs***(useriid, appiid)*

Returns a hash containing all of the appUser's attributes for which the requesting entity has read privilege.

5.6 Retrieving Group Information in Perl

is_group

*$docdir->***is_group***(iid)*

Returns true if the iid corresponds to a **docGroup** entry in the directory.

get_group_attrs

*$docdir->***get_group_attrs***(groupiid)*

Returns a hash containing all of the group's attributes for which the requesting entity has read privilege.

get_group_member_count

*$docdir->***get_group_member_count***(groupiid[, recurse])*

Returns the number of members in the group.

If recurse is true, causes the method to count members of the group and subgroups. False causes only immediate members to be counted. If `recurse` is not specified, it is true by default.

get_group_members

*$docdir->***get_group_members***(groupiid[, recurse])*

Returns an array containing the IIDs of the members of the group. A member corresponds to a **docPerson**, not a subgroup of the group. The keys of the array are numeric and arbitrarily assigned.

If recurse is true, causes the method to return members of the group and all subgroups. False causes only immediate **docPerson** members to be returned. If `recurse` is not specified, it is true by default.

is_group_within_group

*$docdir->***is_group_within_group***(groupiid, supergroupiid[, recurse])*

Returns true if the group is hierarchically contained by the supergroup, false otherwise.

If `recurse` is true, the method examines the group and all subgroups for membership. False causes the method to consider only immediate member groups. If `recurse` is not specified, it is true by default.

get_parent_group

*$docdir->***get_parent_group***(groupiid)*

Returns the IID of the immediate parent of the group. If the group is the highest in its hierarchy, returns an empty string.

get_subgroups

*$docdir->***get_groups***(groupiid[, recurse])*

Returns an array containing the IIDs of the immediate subGroups of the group. If the group is the lowest in its hierarchy, returns an empty array. The keys of the array are numeric and arbitrarily assigned.

If `recurse` is true, the method returns all member groups and subgroups. False causes only immediate member groups to be returned. If `recurse` is not specified, it is true by default.

6 The Directory API for PHP

6.1 The docdirectory Object

To use the Directory API in PHP, one creates a DocDirectory object and calls methods on it.

The DocDirectory object's handle is obtained by calling the `new` method of the DocDirectory class. The `new` method takes the name of the *docDirectory.conf* file as its only parameter.

Once the DocDirectory handle is obtained, a variety of methods are available to obtain information from the directory service and Group Service.

6.2 Obtaining a docdirectory Handle

new

```
$docdir = new docdirectory(docdirectoryconffilename)
```

Constructor. Creates a new docdirectory object handle initialized with the parameters of the *docDirectory.conf* file. The *docdirectoryconffilename* parameter may include the file's path if necessary.

6.3 Retrieving App Information in PHP

isApp

```
$docdir->isApp(iid)
```

Returns true if the iid corresponds to a **docApp** entry in the directory.

getAppAttrs

```
$docdir->getAppAttrs(appiid)
```

Returns an array containing all of the application's attributes for which the requesting entity has read privilege. The keys of the array are the names of the attributes.

6.4 Retrieving Person Information inPHP

isPerson

```
$docdir->isPerson(iid)
```

Returns true if the iid corresponds to a **docPerson** entry in the directory.

getIidForUname

```
$docdir->getIidForUname(username)
```

Returns the IID of the user.

getPersonAttrs

*$docdir->***getPersonAttrs***(useriid)*

Returns an array containing all of the user's attributes for which the requesting entity has read privilege. The keys of the array are the names of the attributes.

getPeopleAttrs

*$docdir->***getPeopleAttrs***(useriids, attributes)*

Returns an array filled with associative arrays. Each associative array contains one **docPerson**'s attributes for which the requesting entity has read privilege. If any item in the `useriids` array does not correspond to a **docPerson** entry, it is ignored.

Takes the following arguments:

useriids

An array containing the IIDs that identify **docPerson** entries.

attributes

An array containing the attributes that are to be returned. It is not necessary to include `docIid` in this list, as it is always returned.

isPersonInGroup

*$docdir->***isPersonInGroup***(useriid, groupiid, recurse)*

Returns true if the person is a member of the group, false otherwise. If the recurse parameter is true, the method examines all subgroups for membership. False causes the method to consider only direct membership in the group.

getGroupsForPerson

*$docdir->***getGroupsForPerson***(useriid, recurse)*

Returns an array containing the IIDs of groups in which the user is a member. The keys of the array are numeric and arbitrarily assigned. If `recurse` is true, the method returns all supergroups. False causes the method to return only groups in which the person has direct membership.

getSubgroupsForPerson

*$docdir->***getSubgroupsForPerson***(useriid, supergroupiid)*

Returns an array containing the IIDs of all subgroups that are hierarchically under the supergroup in which the user is a member. The keys of the array are numeric and arbitrarily assigned.

6.5 AppUser Information

6.5.1 Retrieving AppUser Information in PHP

getAppuserAttrs

*`$docdir->`***`getAppuserAttrs`***`(useriid, supergroupiid)`*

Returns an array containing all of the appUser's attributes for which the requesting entity has read privilege. The keys of the array are the names of the attributes.

6.5.2 Setting AppUser Information in PHP

setAppuserAttrs

*`$docdir->`***`setAppuserAttrs`***`(useriid, appiid)`*

Returns an array containing all of the appUser's attributes for which the requesting entity has read privilege. The keys of the array must be the names of the attributes.

6.6 Retrieving Group Information in PHP

isGroup

*`$docdir->`***`isGroup`***`(iid)`*

Returns true if the iid corresponds to a **docGroup** entry in the directory.

getGroupAttrs

*`$docdir->`***`getGroupAttrs`***`(groupiid)`*

Returns an array containing all of the group's attributes for which the requesting entity has read privilege. The keys of the array are the names of the attributes.

getGroupMemberCount

*`$docdir->`***`getGroupMemberCount`***`(groupiid, recurse)`*

Returns the number of members in the group.

If recurse is true, causes the method to count members of the group and subgroups. False causes only immediate members to be counted.

getGroupMembers

*$docdir->***getGroupMembers***(groupiid, recurse)*

Returns an array containing the IIDs of the members of the group. A member corresponds to a **docPerson**, not a subgroup of the group. The keys of the array are numeric and arbitrarily assigned.

If `recurse` is true, causes the method to return members of the group and all subgroups. False causes only immediate **docPerson** members to be returned.

isGroupWithinGroup

*$docdir->***isGroupWithinGroup***(groupiid, supergroupiid, recurse)*

Returns true if the group is hierarchically contained by the supergroup, false otherwise.

If `recurse` is true, the method examines the group and all subgroups for membership. False causes the method to consider only immediate member groups.

getParentGroup

*$docdir->***getParentGroup***(groupiid)*

Returns the IID of the immediate parent of the group. If the group is the highest in its hierarchy, returns an empty string.

getSubgroups

*$docdir->***getGroups***(groupiid, recurse)*

Returns an array containing the IIDs of the immediate subGroups of the group. If the group is the lowest in its hierarchy, returns an empty array. The keys of the array are numeric and arbitrarily assigned.

If `recurse` is true, the method returns all member groups and subgroups. False causes only immediate member groups to be returned.

Figure 1: Directory Subsystem Diagram

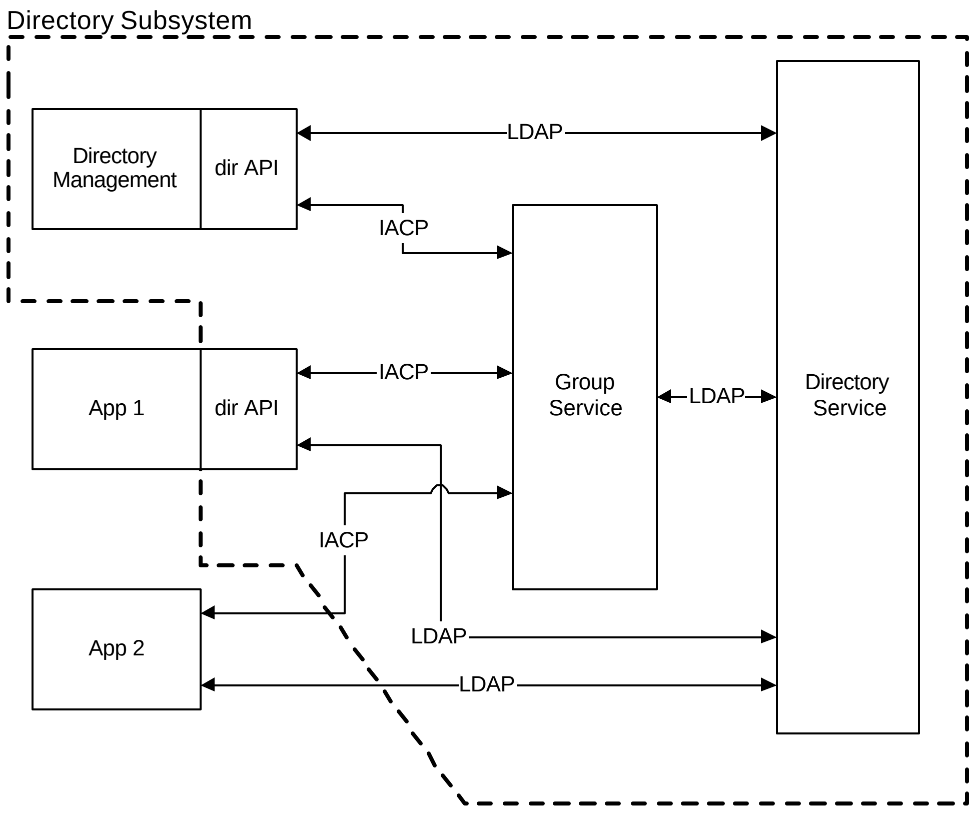
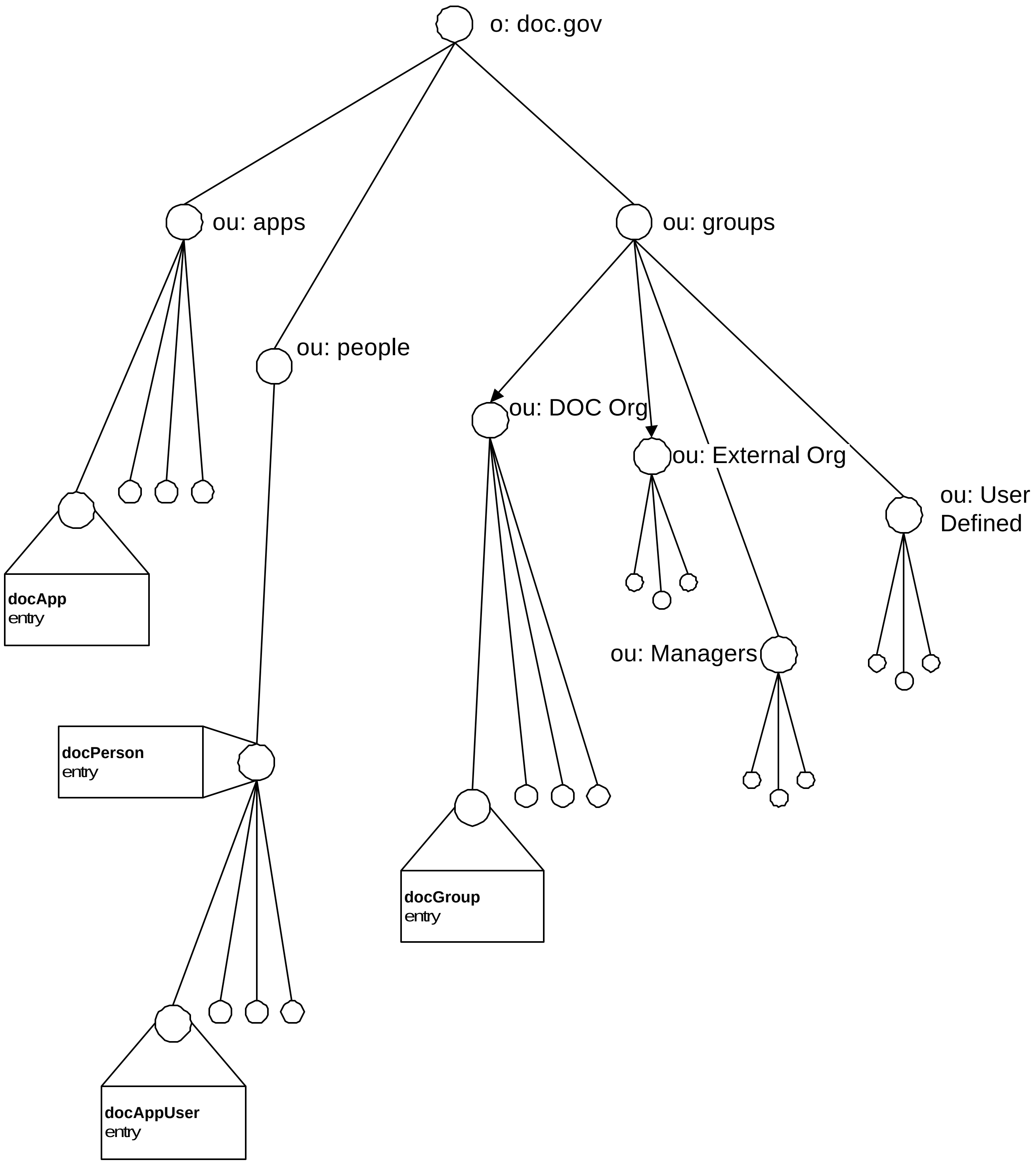


Figure 2: Directory Schema Diagram



Appendix F: DOC Intranet Inter-Application Communication Protocol

1 Introduction

This document defines the Department of Commerce Inter-Application Communication Protocol (DOCIACP), by which participating applications within the Department of Commerce Intranet can communicate with each other. Communications are carried on top of the HTTP protocol. As such, the communications consist of exactly two participants, referred herein as the client and the server. The client is the participant which initiates the transaction, by requesting information from the server.

The goals of the DOCIACP include the following:

- The server should be able to positively authenticate the client's identity
- The server should be able to determine, based on the client's identity, what, if any, information to provide to the client
- The server should have the option of requiring an encrypted (SSL) link
- Information exchange should be in XML when practicable, to provide ease of interchange and manipulation. However, this should not exclude other content-types from being returned by the servers when appropriate.
- Both clients and servers should be easy to implement in a number of languages, including at the minimum, Java, Perl and PHP
- There should be a well defined interface for a client to determine what services are available from the server, and what the calling convention is to receive those services

The DOCIACP protocol makes use of public key cryptography methods to accomplish its authentication tasks. See [CRYPTO] for details.

This document does not cover the various issues related to storage and retrieval of public and private keys, though it is expected that public keys will be stored in a directory service such a defined by [DIRSVC] or as provided by a third party PKI implementation..

This document does not cover the Applications Programming Interfaces (API's) used on either the client or server side to build applications. See [IACP-API] for specific information on API's and usage.

2 Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]. An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be "conditionally compliant."

The Inter-Application Communication Protocol provides a means of communication between two participants, the client and the server. The communication is referred to herein as the transaction.

2.1 DOCIACP Client Application Requirements

The client application MUST support communications with HTTP servers as described in [RFC2616].

The client MAY support HTTPS client side communications, as defined in [SSL].

The client MAY support encryption and decryption as defined in [CRYPTO].

Clients supporting encryption and decryption MUST have a public key / private key pair as defined in [CRYPTO].

2.2 DOCIACP Server Application Requirements

The server application MUST support HyperText Transfer Protocol (HTTP), as described in [RFC2616].

The server application MAY support HTTPS, as described in [SSL].

The server application MAY support encryption and decryption as defined in [CRYPTO].

3 Terminology

APPLICATION_IID	The name of the CGI variable which holds an application instance’s unique identifier.
client application	A system which requests data from a server application via HTTP(S)
HTTP header	A header field in a HTTP-response. See [RFC2616]
HTTP CGI parameter	A URL-encoded or form encoded variable in a HTTP request. See [RFC2616]
nonce	A server generated string containing some random component. See Section 6.0 and [RFC2617].
server application	A system that provides a service to client applications HTTP(S)

service	A resource returned by a server in response to a HTTP(S) request
service name	A string which identifies a service provided by a DOCIACP server.
service URL	The URL by which the server is accessed for DOCIACP communications.

4 Overview

The DOCIACP protocol establishes a means of communications between a server and a client within the Department of Commerce Intranet. Participating applications can be clients, servers or both. For any given transaction, an application can be a client or server. If bi-directional exchange of information is required, then multiple transactions must be executed.

Each server provides one or more services to clients. A service is simply an available transaction by which the client makes a request for data. The client forms a request including the service name and any other necessary HTTP CGI parameters to define the request, and the server returns some amount of data in response to that request.

All services for a given server are accessed through a single URL known as the application's DOCIACP service URL. All requests must be via the HTTP POST or GET methods. Since all services are accessed via a single URL, various CGI variables are set to indicate which service is desired. For example, to access the catalog service, a CGI variable 'Service=Catalog' would be submitted with the request. The method of publishing these well-known application specific DOCIACP service URL's is beyond the scope of this document, but it is envisioned that they will be published in the directory service.

Every server providing DOCIACP services is required to provide at least one service, the 'catalog service.' The catalog service is simply a listing of all the available services provided by a server. The catalog is returned as an XML document, conforming to a predefined catalog service DTD. The catalog service will list all services (including itself) by service name, along with the CGI parameters it accepts, and miscellaneous information such as which parameters are optional, and descriptions of the service itself and each of its parameters. Developers should refer to the catalog service on servers they intend to make client connections to so as to ensure that the services and calling parameters are as expected.

Applications can be written in such a way that administrators can configure which services are turned on and which are turned off. The catalog service SHOULD correctly reflect only the currently available services. The server MUST return a graceful 400 class error code if a client requests a service that does not exist or is disabled.

Once a client decides to request a service from a server, the client prepares a HTTP POST request, sets the service name and any other required HTTP CGI parameters, and sends the request to the server.

At this point, the server may optionally require authentication of the client. See section 5.2.

Once the server's authentication and authorization requirements have been met, the server examines the request, and prepares the document to return to the client. This document would usually be an XML document, though it could be anything, including image files or word processing documents. As in HTTP, the type of the return document is communicated back to the client via the Content-type HTTP header.

5 Protocol Definition

The DOCIACP protocol is built on top of HTTP 1.0 protocol. A new authorization scheme is introduced to provide public key based authentication. As part of this authentication scheme, each instance of a DOCIACP application has a unique identifier called the APPLICATION_IID. The APPLICATION_IID is not tied to a particular piece of software, but rather, to a particular installed and running copy of the software. This allows servers to decide with which applications they wish to share data on a fine-grained basis. For example, a given server may wish to only provide services to the installed copy of a client application designated to handle a particular Bureau.

All applications, whether acting as client or server, require a unique APPLICATION_IID. If an application serves as both a client and a server, then the same APPLICATION_IID is used in both situations. The term APPLICATION_IID may be qualified as a CLIENT_APPLICATION_IID or a SERVER_APPLICATION_IID for purposes of clarifying which application’s APPLICATION_IID is referred to. This in no way indicates that a single application has more than one APPLICATION_IID.

5.1 The DOCIACP Request

A transaction begins with a client constructing a HTTP 1.0 request for a service. The connection is made to a well-known service URL for the server. The following is an example request for the catalog service from a server application.

```
POST /cgi-bin/IACP_URL HTTP/1.0

Content-type: text/x-www-form-urlencoded

Accept: text/xml



Service=Catalog

Source=<CLIENT_APPLICATION_IID>
```

Table 1 summarizes the service request format. Note that the required? column refers to whether it is a protocol violation to not return that item, not to whether or not the server will have sufficient information to complete the request. For example, it is not a protocol violation to not supply an Authorization header if so requested, but the server will almost certainly not return the information desired by the client.

Table 1: Minimum DOCIACP Client Request Components

Item Type	Name	Value	Required?
HTTP Request	POST	<path to handler> HTTP/1.0	YES
HTTP Header	Content-type:	text/x-www-form-urlencoded	YES
HTTP Header	Authorization:	DOCIACP <encoded nonce>	NO
CGI Variable	Service	<Service Name>	YES
CGI Variable	Source	<CLIENT_APPLICATION_IID>	YES

See Section 5.2 for discussion of the Authorization header.

In addition to the components listed in Table 1, clients can supply additional headers and CGI variables as defined by the HTTP specification, and as required as parameters to the particular service being requested. In particular, clients may wish to include headers relating to cache and proxy control. The server may or may not choose to use additional components.

5.2 The DOCIACP Authentication Scheme

On receiving a request, the server may either service the request or demand authentication information. Servers not requiring authentication merely construct a response, as per section 5.3.

The authentication process is per the HTTP 1.0 specification, with a new authentication type. Servers requiring authentication return a 'WWW-Authenticate DOCIACP_AUTH' line to the client indicating that authentication is required. The DOCIACP_AUTH term denotes the type of authorization required, and is the only authorization type defined by this protocol. Servers and clients MAY also support BASIC or DIGEST authentication schemes per [RFC2617].

The WWW-Authenticate header returned to the client also contains a 'nonce' (See [RFC2617] for a definition). A nonce is a pseudo-random string of characters which is used as a challenge string. When a client receives a nonce as part of a WWW-Authenticate HTTP header, the client is responsible for encrypting the nonce with the client's own private key, and returning the encrypted nonce to the server in a Authorization header. Once the server receives the Authorization header from the client, the server can verify the client's identity by decrypting the nonce using the client's public key. Servers operating in non-SSL mode should consider utilizing a nonce generation policy which guards against replay attacks. See section 6.0 for details.

The following transcript describes a DOCIACP_AUTH transaction.

As above, the transaction begins with the client requesting access to a service.

```
POST /cgi-bin/IACP_URL HTTP/1.0
```

```
Content-type: text/x-www-form-urlencoded
Accept: text/xml


Service=Catalog
Source=<CLIENT_APPLICATION_IID>
```

This time, the server demands authentication.

```
HTTP/1.0 401 Unauthorized
Date: Fri, 31 Dec 1999 23:59:59 GMT
Server: <SERVER_APPLICATION_IID>
Content-type: text/plaintext
WWW-Authenticate: DOCIACP_AUTH aJq237YYYzp034rgPOkj
```

Here, the nonce is ' aJq237YYYzp034rgPOkj'. Upon receiving this, the client encrypts the nonce using the client's private key, and resends the request, repeating back the plaintext nonce in a Nonce header, and adding an Authorization header.

```
POST /cgi-bin/IACP_URL HTTP/1.0
Content-type: text/x-www-form-urlencoded
Accept: text/xml
Nonce: aJq237YYYzp034rgPOkj
Authorization: 3!h&h2)!z94e]@62


Service=Catalog
Source=<CLIENT_APPLICATION_IID>
```

On receiving the Authorization HTTP header, the server must look up the client’s public key, and verify the authenticity of the client by decrypting the nonce using the key. If the nonce decrypts correctly, then the client is authenticated. The server must then decide what, if any, data to provide to the client.

5.3 The DOCIACP Response

If the server decides to service the request, it simply returns a page in accordance with the HTTP 1.0 specification. No additional headers are required. An example response follows:

```
HTTP/1.0 200 OK

Date: Fri, 31 Dec 1999 23:59:59 GMT

Content-type: text/xml



<?xml version="1.0"?>

<!DOCTYPE SYSTEM
"http://dtd.doc.gov/iacp_catalog.dtd">

<Catalog>
     <Service>

          <ServiceName>Catalog</ServiceName>

          <ServiceDescription>

               Returns XML describing all

               available services

          </ServiceDescription>

          <ReturnType>text/xml</ReturnType>

     </Service>

</Catalog>
```

Here, the server has returned a catalog containing only itself as a service.

6 Selection of Nonce values

Per [RFC2617], the nonce is a server-specified data string which should be uniquely generated each time a 401 response is made. It is recommended that this string be base64 or hexadecimal data. Specifically, since the string is passed in the header lines as a quoted string, the double-quote character is not allowed.

The contents of the nonce are implementation dependent. The quality of the implementation depends on a good choice.

Transactions which are not conducted over a secure SSL link are potentially susceptible to replay attacks, in which third parties listening in on the transaction capture the Authorization header, and resubmit the request using the stale Authorization header. A number of methods of nonce selection are possible to minimize or eliminate the risks of such a replay attack.

First, replay attacks are not possible over SSL links. Servers wanting maximum protection of their resources should make resources available only over SSL links.

Second, a timestamp can be encoded into the nonce. Authorization headers referencing a given nonce are only accepted within some fixed amount of time beyond the timestamp contained in the nonce. Since the nonce is encrypted with the client's private key, third parties could only use the nonce as stamped, limiting unauthorized disclosure.

Third, the client's IP address could be encoded into the nonce. This requires third parties to both intercept the transaction, and successfully IP-spoof their address. Encoding the client's IP address into the nonce looses its effectiveness when proxies are involved, as spoofing becomes unnecessary when the third party is able to go through the same proxy server as the client.

Finally, the server can maintain a list of nonces it has given out, and remove a nonce from the list when it is first submitted by any client. No other third party could not then replay the Authorization header, as the nonce it refers to would no longer be valid. This technique eliminates the possibility of replay attacks, at the cost of the server having to save generated nonce values across consecutive connections.

The method used to generate the nonce as well as the server's policy for nonce acceptance is opaque to the client. Servers providing non-SSL encrypted service and wanting maximum security SHOULD implement a one-time nonce policy.

7 Catalog Service Definition

The catalog service is the only service which all DOCIACP servers must implement. The catalog service produces a xml document conforming to the catalog service DTD. The DTD is as follows

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT Catalog (Service+)>

<!ELEMENT Service (

     ServiceName,

     ServiceDescription,

     ReturnType?,

     ReturnTypeDTD?,
     Parameter*)>

<!ELEMENT Parameter (

     ParamName,

     ParamDescription)>

<!ELEMENT ServiceName (#PCDATA)>

<!ELEMENT ServiceDescription (#PCDATA)>

<!ELEMENT ReturnType (#PCDATA)>

<!ELEMENT ReturnDTD (#PCDATA)>

<!ELEMENT ParamName (#PCDATA)>

<!ELEMENT ParamDescription (#PCDATA)>
```

As specified in the DTD, the catalog service must return an XML document which includes at least one service – the catalog service. Each service consists minimally of a Service Name and a Service Description. The Service Name is the parameter used by clients to request the service.

An example Catalog Service XML document follows.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE Catalog SYSTEM
"http://dtd.doc.gov/iacp_catalog.dtd">

<Catalog>

     <Service>

          <ServiceName>Catalog</ServiceName>

          <ServiceDescription>

               Listing of all available services

               from this DOCAIP server

          </ServiceDescription>

          <ReturnType>text/xml</ReturnType>

          <ReturnDTD>
```

```
                    http://dtd.doc.gov/iacp_catalog.dtd

               </ReturnDTD>

          </Service>

          <Service>

               <ServiceName>GetDateAsString</ServiceName>

               <ServiceDescription>

                    Returns a Date-Time stamp as a 
string.

                    EST is assumed, set param 
CGI=<timezone>

                    to override

               </ServiceDescription>

               <ReturnType>text/plaintext</ReturnType>

               <Parameter>

                    <ParamName>Timezone</ParamName>

                    <ParamDescription>

                         The time zone in which you wish 
                    the

                         returned string to be displayed.

                    </ParamDescription>

               </Parameter>

     </Catalog>
```

Here, the server identifies two services it provides. The first is of course the obligatory catalog service itself. The second is a time of day service.

The time service has a service name of GetDateAsString. Clients wishing to access this service must pass in a HTTP CGI parameter Service=GetDateAsString.

The service returns a plaintext representation of a time-date stamp.

The service has a single parameter, named Timezone. Clients wishing to see the time formatted to, for example, Pacific Standard Time will provide a HTTP CGI parameter Timezone=PST.

8 Issues

Should clients be able to authenticate the server in non- SSL mode by sending its own nonce in the reqeust.

Should we expand the catalog service to delineate the allowable types and values for service parameters.

9 References

[CRYPTO].	DOC Intranet Cryptographic Infrastructure
[DIRSVC]	Directory Schema for the DOC Intranet Directory Service
[IACP-API]	DOC Intranet IACP API and Tutorial
[RFC2119]	Key words for use in RFCs to Indicate Requirement Levels (RFC 2119), available http://www.faqs.org/rfcs/rfc2119.html
[RFC2616]	Hypertext Transfer Protocol -- HTTP/1.1 (RFC 2616), available http://www.faqs.org/rfcs/rfc2616.html
[RFC2617]	HTTP Authentication: Basic and Digest Access Authentication, available http://www.faqs.org/rfcs/rfc2617.html
[SSL]	The SSL Protocol, Version 3.0, available http://home.netscape.com/eng/ssl3/ssl-toc.html

Apendix G: DOC Inter-Application Communication Protocol API

Version 1.0

Revised: 2000.11.12

1 Introduction

Applications within the DOC intranet can share information via the DOC Inter-Application Communication Protocol (IACP). The purpose of the API is to provide an easy means for programmers to supply and obtain information via IACP. This document describes APIs for Java, Perl, and PHP.

2 Terminology

DOC

Department of Commerce.

IACP

Inter-Application Communication Protocol. The DOC protocol that this API facilitates. See [IACP].

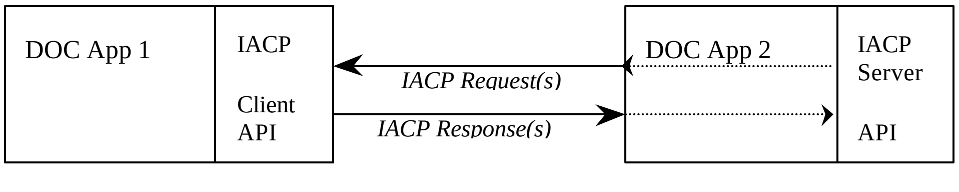
3 Overview

3.1 Using the IACP APIs

The IACP API provides applications an easy means of supplying and obtaining information via IACP. The IACP library methods handle protocol details and perform networking so the developer can concentrate on the application they are implementing rather than its environment infrastructure.

An application supplying information via IACP uses the IACP Server API, and an application that needs to obtain information via IACP uses the IACP Client API. If an application needs to do both, it will use both.

The below diagram depicts use of the IACP API .



The application supplying the IACP service (DOC App 2) shows intermediate arrow to the IACP Server API because it delegates requests to the IacpHandler rather than providing for it to handle them directly.

4 The IACP Client API for Java

The IACP client API for Java is in package gov.doc.intranet. The primary class is **IacpClient**. You can discover an application's services via the `getCatalog` method. It returns an array of **IacpService** objects. From these you can obtain the names and details of the application's services. If you want to request from one of these services, the **IacpClient** provides `getResponse()` methods. The **IacpClient** handles whatever authentication is demanded by the service transparently.

4.1 The IacpClient Class

An IacpClient object is used to request information from other applications via IACP. The following depicts a typical use of an IacpClient.

```
IacpClient myClient = new IacpClient(MY_IID, MY_PRIVATE_KEY);

myClient.setServiceUrl(theServiceUrl);


/*************************************************************/

/* For a binary contents you can use IacpResponse directly. */

/*************************************************************/

IacpResponse myResponse;

Properties paramProperties;

// set appropriate service parameters in paramProperties.


try {

    myResponse =

        myClient.getResponse("aBinaryService", paramProperties);

} catch (IOException exc) {

    // handle it

} catch (AuthenticationException exc) {

    // handle it
}

/* Now get the bytes for use. */
byte[] myByteArray = myResponse.getBytes();

/*********************************************************/
/* For other return types, use a subclass of IacpResponse. */
/*********************************************************/

/* Get a StringIacpResponse. */
// set appropriate service parameters in paramProperties.

try {
    myResponse =
        myClient.getResponse("aStringService", paramProperties);
} catch (IOException exc) {
    // handle it
} catch (AuthenticationException exc) {
    // handle it
}

/* Create the StringIacpResponse and get the String. */
StringIacpResponse myStringResponse;
try {
    myStringResponse = new StringIacpResponse(myResponse);
} catch(BadContentTypeException) {
    // handle it
}
```

```
String myReturnString = myStringResponse.getString();

/* Get an XmlIacpResponse. */
// set appropriate service parameters in paramProperties.

try {
    myResponse =
        myClient.getResponse("anXmlService", paramProperties);
} catch (IOException exc) {
    // handle it
} catch (AuthenticationException exc) {
    // handle it
}

/* Create the XmlIacpResponse and get the XML document. */
XmlIacpResponse myXmlResponse;
try {
    myXmlResponse = new XmlIacpResponse(myResponse);
} catch(BadContentTypeException) {
    // handle it
}

try {
    org.w3c.dom.Document myDocument = myXmlResponse.getDocument();
} catch (...Exception) {
    // handle it
}
```

Signature

public class **IacpClient** extends Object

Constructors

IacpClient

```
public IacpClient()
```

IacpClient

```
public IacpClient(String myIid,
                  java.security.PrivateKey myPrivateKey)
```

Parameters:

myIid - My application IID.

myPrivateKey - The private key for my application. The IacpClient uses the private key to use a service that requires authentication.

Methods

setIid

```
public void setIid(String myIid)
```

Parameters:

myIid - My application IID.

---

setPrivateKey

```
public void setPrivateKey(PrivateKey myPrivateKey)
```

Parameters:

myPrivateKey - The private key for my application. The IacpClient uses the private key to use a service that requires authentication.

---

setServiceUrl

```
public void setServiceUrl(URL serviceUrl)
```

Parameters:

serviceUrl - The URL of the IACP service from whom information will be requested.

---

getCatalog

```
public IacpService[] getCatalog()
```

Returns:

An array of IacpService objects. Each IacpService object describes a service available from an application providing IACP services.

---

getResponse

```
public IacpResponse getResponse(String serviceName)
```

Use this method if you don't need to send parameters to an IACP service.

Parameters:

serviceName - The name of the IACP service to be requested.

Returns:

An IacpResponse object. Unless the IacpResponse content is binary, it will probably be used to initialize one of its subclasses, such as XmlIacpResponse or StringIacpResponse.

---

getResponse

```
public IacpResponse getResponse(String serviceName,
                                Properties params)
```

Parameters:

serviceName - The name of the IACP service to be requested.

params - The parameters to be used as input to the IACP service.

Returns:

An IacpResponse object. Unless the IacpResponse content is binary, it will probably be used to initialize one of its subclasses, such as XmlIacpResponse or StringIacpResponse.

4.2 The IacpResponse Class

The IacpResponse object is used to obtain the information in the reply from an IACP service request. It is returned by the IacpClient.getResponse method. If the information in the reply is binary in nature, the IacpResponse.getBytes() method can be used to obtain the information in raw binary form.

Subclasses of IacpResponse exist to facilitate obtaining a response of another content-type. If your application wishes to provide a response in some proprietary or encoded form, you can create a subclass of IacpResponse to help the consumer of of your IACP service information.

The following demonstrates the use of IacpResult for obtaining binary data.

```
IacpResponse myResponse;

Properties paramProperties;

// set appropriate service parameters in paramProperties.


try {

    myResponse =

        myClient.getResponse("aBinaryService", paramProperties);

} catch (IOException exc) {

    // handle it

} catch (AuthenticationException exc) {

    // handle it

}


/* Now get the bytes for use. */

byte[] myByteArray = myResponse.getBytes();
```

The following demonstrates use of the XmlIacpResult subclass of IacpResult.

```
// set appropriate service parameters in paramProperties.

try {
    myResponse =
        myClient.getResponse("anXmlService", paramProperties);
} catch (IOException exc) {
    // handle it
} catch (AuthenticationException exc) {
    // handle it
}

/* Create the XmlIacpResponse and get the XML document. */
XmlIacpResponse myXmlResponse;
try {
    myXmlResponse = new XmlIacpResponse(myResponse);
} catch(BadContentTypeException) {
    // handle it
}

try {
    org.w3c.dom.Document myDocument = myXmlResponse.getDocument();
} catch (...Exception) {
    // handle it
}
```

Signature

public class **IacpResponse** extends Object

Variables

---

contentType

```
protected String contentType
```

---

content

```
protected byte content[]
```

Constructors

IacpResponse

```
protected IacpResponse()
```

IacpResponse

```
protected IacpResponse(String contentType,
                       byte content[])
```

Parameters:

contentType - The value of the content-type header field of the IACP service response.

content - The content of the IACP service response.

Methods

---

getContentType

```
public String getContentType()
```

Returns:

The value of the content-type header field of the IACP service response.

---

getContent

```
public byte[] getContent()
```

Returns:

The content of the IACP service response.

4.4 The XmlIacpResponse Class

The XmlIacpResponse object extends IacpResponse and adds the getXmlContent() method.

Signature

public class **XmlIacpResponse**

extends IacpResponse

Constructors

XmlIacpResponse

```
public XmlIacpResponse()
```

XmlIacpResponse

```
public XmlIacpResponse(IacpResponse response) throws BadContentTypeException
```

Parameters:

response - The IacpResponse obtained from the IacpClient.getResponse() method.

Throws: BadContentTypeException

If the content-type of the response is specified and is not "text/xml".

Methods

---

setResponse

```
public void setResponse(IacpResponse response)
```

> **Throws:** BadContentTypeException
>
> If the content-type of the response is specified and is not "text/xml". Note that the contents are set before the Exception is thrown, so it is still possible to call the getXmlContent() method, which you would expect to throw a parsing Exception. This is included as a convenience in case the IACP service is improperly implemented.

---

getXmlContent

```
public Document getXmlContent() throws IOException
```

> **Returns:**
>
> The XML response in the form of a Document object.
>
> **Throws:** IOException
>
> This is just a placeholder in the current XmlIacpResponse specification. The actual exceptions thrown will be implementation dependent.

4.5 The StringIacpResponse Class

The StringIacpResponse object extends IacpResponse and adds the getStringContent() method.

Signature

public class **StringIacpResponse** extends IacpResponse

Constructors

StringIacpResponse

```
public StringIacpResponse()
```

StringIacpResponse

```
public StringIacpResponse(IacpResponse response) throws BadContentTypeException
```

Parameters:

response - The IacpResponse obtained from the IacpClient.getResponse() method.

Throws: BadContentTypeException

If the content-type of the response is specified and is not "text/plain".

Methods

---

setResponse

```
public void setResponse(IacpResponse response)
```

Throws: BadContentTypeException

If the content-type of the response is specified and is not "text/plain". Note that the contents are set before the Exception is thrown, so it is still possible to call the getStringContent() method to obtain the results as a String. The normal way to do this given that the content-type is not not specified as "text-plain", however, would be to convert the byte[] obtained from IacpResult.getContent() to a String. This is included as a convenience in case the IACP service is improperly implemented.

---

getStringContent

```
public String getStringContent()
```

Returns:

The response in the form of a String.

4.6 The IacpService Class

The IacpService object describes an IACP service provided by an application. An array of IacpService objects is returned by the IacpClient.getCatalog() method. Each object is the application's description of a single IACP service it offers.

Signature

public class **IacpService** extends Object

Variables

---

name

```
protected String name
```

---

description

```
protected String description
```

---

returnType

```
protected String returnType
```

---

returnDtd

```
protected String returnDtd
```

---

parameters

```
protected Properties parameters
```

Constructors

IacpService

```
protected IacpService()
```

IacpService

```
protected IacpService(String name,
                    String description)
```

IacpService

```
protected IacpService(String name,
                    String description,
                    String returnType,
                    String returnDtd,
                    Properties parameters)
```

Methods

getName

```
public String getName()
```

Returns:

The name of the service.

getDescription

```
public String getDescription()
```

Returns:

A description of the service.

getReturnType

public String getReturnType()

Returns:

The returnType of the service. Note that returnType is optional to the catalog entry. If it is not specified, this method returns null.

---

getReturnDtd

public String getReturnDtd()

Returns:

The ReturnDTD of the service. Note that this is the string returned by the catalog: it has not been checked for existence or validity. Note also that returnDTD is optional to the catalog. If it is not specified, this method returns null.

---

getParameters

public Properties getParameters()

Returns:

The parameters accepted by the service as input. Note that parameters are optional to the service. If the service does not accept parameters, this method returns null.

4.7 The BadContentTypeException Class

Thrown by a subclass of IacpResponse when the content-type of the response is found to be inappropriate for the subclass' use in processing it.

Signature

public class **BadContentTypeException** extends IOException

Constructors

BadContentTypeException

public BadContentTypeException()

Constructs a BadContentTypeException with no specified detail message.

BadContentTypeException

public BadContentTypeException(String msg)

Constructs a BadContentTypeException with the specified detail message.

---

5 The IACP Server API for Java

The IACP server API for Java is also in package gov.doc.intranet. The primary class is **IacpHandler**. You initialize an **IacpHandler** via the `examineRequest` method. You can then get the parameters and relevant headers of the request, set the authentication scheme and check for authorization, and use the **IacpHandler** to return a variety of failure responses.

5.1 The IacpHandler Class

An IacpHandler object is used to process requests from other applications via IACP. The following depicts a typical use of an IacpHandler.

```
IacpHandler myHandler = new IacpHandler(docDirectory);

myHandler.examineRequest(httpReq, httpRes);


/* This app may want to restrict access or customize response based

 * on the identity of the requesting app.

 */

String reqIid = myHandler.getRequesterIid();


/* We can request paramaters individually by name or all at once. */

Properties parameters = myHandler.getParameters();


String reqService = myHandler.getReqestedService();

if (reqService.equals("catalog")) {

    /* We have decided not to require authentication for the catalog
```

```
         * service.
         */
        myHandler.setAuthenticationScheme(AUTH_NONE);
        /* note that myCatalogGenerationMethod contains a
         * httpRes.setContentType() call that sets the response
         * content-type to "text/xml".
         */
        myCatalogGenerationMethod(...);
        return;
    }

    /* The rest of our services use AUTH_IACP_TIME_STAMP, so we can check
     * authentication here rather than individually per service.
     */
    myHandler.setAuthenticationScheme(AUTH_IACP_TIME_STAMP);
    if (!myHandler.isAuthorized()) {
        /* HTTP response code 401 */
        myHandler.demandAuthentication();
        return;
        }

    if (reqService.equals("StringService1")) {
         /* The catalog service has provided a ReturnType for this service of
          * "text/plain".  The myStringService1ResponseMethod contains a
          * httpRes.setContentType() call that sets the response content-type
          * to "text/plain".
          */
         myStringService1ResponseMethod(...);
         return;
    } else if (reqService.equals("XmlService2")) {
         /* The catalog service has provided a ReturnType for this service of
          * "text/xml".  The myXmlService2ResponseMethod contains a
          * httpRes.setContentType() call that sets the response content-type
          * to "text/xml".
          */
         myXmlServiceResponse2Method(...);
         return;
    } else {
        /* HTTP response code 404 */
        myHandler.returnNotFoundError();
    }
```

Signature

public class **IacpHandler** extends Object

Variables

AUTH_NONE

public static final int AUTH_NONE

Authentication scheme: Instructs the IacpHandler to ignore authentication.

AUTH_BASIC

public static final int AUTH_BASIC

Authentication scheme: Instructs the IacpHandler to use Basic authentication as described in [RFC 2617].

AUTH_DIGEST

public static final int AUTH_DIGEST

Authentication scheme: Instructs the IacpHandler to use Digest authentication as described in [RFC 2617].

AUTH_IACP_SIMPLE

public static final int AUTH_IACP_SIMPLE

Authentication scheme: Instructs the IacpHandler to use the IACP authentication as described in the protocol description. Neither time stamp or single-use techniques are to be employed.

AUTH_IACP_TIME_STAMP

public static final int AUTH_IACP_TIME_STAMP

Authentication scheme: Instructs the IacpHandler to include a timestamp in the authentication nonce and check that each returned nonce is no older than the age set by setMaxNonceAge().

AUTH_IACP_SINGLE_USE

public static final int AUTH_IACP_SINGLE_USE

Authentication scheme: Instructs the IacpHandler to keep track of nonces so that it can confirm that each nonce is used only once. This is not likely to be implemented in the IACP version 1.0 library, and may throw an IllegalArgumentException.

AUTH_IACP_SINGLE_USE_TIME_STAMP

public static final int AUTH_IACP_SINGLE_USE_TIME_STAMP

Authentication scheme: Combines AUTH_IACP_SINGLE_USE and AUTH_IACP_TIME_STAMP. This is not likely to be implemented in the IACP version 1.0 library, and may throw an IllegalArgumentException.

Constructors

IacpHandler

```
public IacpHandler(DocDirectory docDirectory)
```

Parameters:

docDirectory - A DocDirectory object.

Methods

---

setMaxNonceAge

```
public void setMaxNonceAge(int age)
```

Specifies the maximum age of a returned authentication nonce. Meaningful only if a time-stamp authentication scheme is used.

Parameters:

age - The maximum acceptable age of the nonce in seconds. If the parameter is not set, it defaults to 60.

---

setAuthenticationScheme

```
public void setAuthenticationScheme(int scheme)
```

Specifies the authentication scheme to be enforced by the handler. Default is AUTH_IACP_TIME_STAMP.

---

examineRequest

```
public void examineRequest(HttpServletRequest req,
                           HttpServletResponse res)
```

Causes the IacpHandler to process a request for IACP service, and supplies it the response object through which it may return a response if it is called upon to do so.

This must be called before any request-processing calls are made. The following methods will throw NotInitializedException if called before a call to examineRequest():

- isAuthorized
- getRequesterIid
- getRequestedService

- getParameter
- getParameters
- demandAuthentication
- returnNoAcessError
- returnNotFoundError

Parameters:

req - The incoming servlet request.

req - The outgoing servlet response.

---

isAuthorized

```
public boolean isAuthorized()
```

Indicates that the request meets the requirements of the specified authentication scheme. At the time that it is called, it checks that the provided Authorization passes the currently specified authentication scheme. For IACP authorization schemes, this includes performing asymmetric decryption.

If the current authentication scheme is AUTH_NONE, the method returns true.

If false is returned, the IacpHandler must provide a response to the requester containing HTTP response code 401 (Unauthorized) and a WWW-Authenticate header specifying the Authorization required of the request. To do so, call IacpHandler's demandAuthentication method.

If a service requires a greater level of authorization than IACP authentication provides, it can use the requester's IID to decide if or what it should provide in the response. Should the request fail the application's requirements even though it provided IACP authentication, call the IacpHandler's returnNoAccessError method.

Returns:

True if a valid Authorization header is present in the request. If the authentication scheme is set to AUTH_NONE, always returns true.

getRequesterIid

public String getRequesterIid()

Returns:

The IID of the requesting application.

getRequestedService

public String getRequestedService()

Returns:

The name of the requested service. For instance, if the catalog service is requested, "catalog".

getParameter

public String getParameter(String paramName)

Parameters:

paramName - The parameter for which a value is sought.

Returns:

The value supplied for the paramName parameter. If the parameter was not included in the request, returns null.

getParameters

public Properties getParameters()

Returns:

A Properties containing the parameters supplied by the requester. If no parameters were supplied by the requester, returns null.

demandAuthentication

public void demandAuthentication()

Respond to the requester that they are required to authenticate to access the requested service. Causes the IacpHandler to respond with HTTP response code 401 (Unauthorized) and the appropriate WWW-Authenticate header given the specified authentication scheme.

returnNoAccessError

public void returnNoAccessError()

Respond to requester that they have successfully authenticated, but are not allowed access to the requested service. IacpHandler responds with HTTP response code 403 (Forbidden) .

returnNotFoundError

public void returnNotFoundError()

Respond to requester that the requested service is not available. IacpHandler responds with HTTP response code 404 (Not Found).

5.2 The NotInitializedException Class

Thrown by IacpHandler when the handler is asked to do work before it has been initialized via the examineRequest method.

Signature

public class **NotInitializedException** extends Object

Constructors

NotInitializedException

public NotInitializedException()

Constructs a NotInitializedException with no specified detail message.

NotInitializedException

public NotInitializedException(String msg)

Constructs a NotInitializedException with the specified detail message.

6 The IACP Client API for Perl

The IACP client API for Perl resides in a module named DOCIntranet.pm. It uses CPAN modules Exception and URI::URL. An implementation of this API will also use an XML module. The most likely candidate is the XML::Parser module. The implementation will also use a module to assist with encryption and key handling.

The primary class is **IacpClient**. You can discover an application's services via the `getCatalog` method. It returns an array of **IacpService** objects. From these you can obtain the names and details of the application's services. If you want to request from one of these services, the **IacpClient** provides `getResponse()` methods. The **IacpClient** handles whatever authentication is demanded by the service transparently.

6.1 The IacpClient Class

An IacpClient object is used to request information from other applications via IACP. The following depicts a typical use of an IacpClient.

```
use Exception;

use URI::URL;

use DOCIntranet;


$myClient = new IacpClient($MY_IID, $MY_PRIVATE_KEY);

$myClient->setServiceUrl($theServiceUrl);

#***********************************************************

# For a binary contents you can use IacpResponse directly. *

#***********************************************************
```

```
# (set appropriate service parameters in %passedParams.)

try {
    $myResponse =
        $myClient->getResponse('aBinaryService', %passedParams);
}

catch DOCIntranet::IacpConnectionException connExc =>
  sub {
       // handle it
       return;
  }

catch DOCIntranet::IacpAuthenticationException authExc =>
   sub {
       # handle it
       return;
   }

# Now get the bytes for use.

$myBytes = $myResponse->getBytes();

#**********************************************************
# For other return types, use a subclass of IacpResponse. *
#**********************************************************

# Get an XmlIacpResponse.
```

```
   # (set appropriate service parameters in %passedParams.)

   try {
       $myResponse =
           $myClient->getResponse('anXmlService', %passedParams);
       $myXmlResponse = new XmlIacpResponse($myResponse);
       $myXmlDocument = $myXmlResponse->getDocument();
   }

   catch DOCIntranet::IacpConnectionException connExc =>
     sub {
          # handle it
          return;
     }

   catch DOCIntranet::IacpAuthenticationException authExc =>
      sub {
          # handle it
          return;
      }
```

Signature

package **IacpClient**;

@ISA = ('UNIVERSAL');

Constructor

`$myIacpClient = new IacpClient([$myIid, $myPrivateKey]);`

Parameters:

myIid - My application IID.

myPrivateKey - The private key for my application. The IacpClient uses the private key to use a service that requires authentication.

Methods

setIid

`$myIacpClient->setIid(myIid);`

Parameters:

myIid - My application IID.

---

setPrivateKey

`$myIacpClient->setPrivateKey(myPrivateKey);`

Parameters:

myPrivateKey - The private key for my application. The IacpClient uses the private key to use a service that requires authentication.

---

setServiceUrl

`$myIacpClient->setServiceUrl($theServiceUrl);`

Parameters:

serviceUrl - The URI::URL of the IACP service from whom information will be requested.

---

getCatalog

`@theServices = $myIacpClient->getCatalog();`

Returns:

An array of IacpService objects. Each IacpService object describes a service available from an application providing IACP services.

---

getResponse

`$myIacpResponse = $myIacpClient->getResponse($serviceName[, %params]);`

Parameters:

serviceName - The name of the IACP service to be requested.

params - The parameters to be used as input to the IACP service. The keys are the param names.

Returns:

An IacpResponse object. Unless the IacpResponse content is binary, it will probably be used to initialize one of its subclasses, such as XmlIacpResponse.

6.2 The IacpResponse Class

The IacpResponse object is used to obtain the information in the reply from an IACP service request. It is returned by the IacpClient.getResponse method. If the information in the reply is binary or a plain string, the IacpResponse->getContent() method can be used to obtain the information in binary form or as a string.

Subclasses of IacpResponse exist to facilitate obtaining a response of another content-type. If your application wishes to provide a response in some proprietary or encoded form, you can create a subclass of IacpResponse to help the consumer of of your IACP service information.

The following demonstrates the use of IacpResult for obtaining binary data.

```
use Exception;

use URI::URL;

use DOCIntranet;


# (set appropriate service parameters in %passedParams.)


try {

    $myResponse =
        myClient->getResponse('aBinaryService', %passedParams);
    $myBytes = $myResponse.getBytes();
}

catch DOCIntranet::IacpConnectionException connExc =>
  sub {
       # handle it
       return;
  }

catch DOCIntranet::IacpAuthenticationException authExc =>
   sub {
       # handle it
       return;
   }
```

The following demonstrates use of the XmlIacpResult subclass of IacpResult.

```
use Exception;
use URI::URL;
use DOCIntranet;

# (set appropriate service parameters in paramProperties.)

try {
    $myResponse =
        $myClient->getResponse('anXmlService', %passedParams);
    $myXmlResponse = new XmlIacpResponse($myResponse);
    $myDocument = $myXmlResponse->getDocument();
}

catch DOCIntranet::XmlParseException authExc =>
    sub {
       # handle it
       return;
    }
```

Signature

package **IacpResponse**;

@ISA = ('UNIVERSAL');

Constructor

`$myResponse = new Response([$contentType, $content]);`

Parameters:

contentType - The value of the content-type header field of the IACP service response.

content - The content of the IACP service response.

Methods

---

getContentType

`$theContentType = $myResponse->getContentType();`

Returns the value of the content-type header field of the IACP service response.

---

getContent

`$theContent = $myResponse->getContent();`

Returns the contents of the IACP service response.

6.3 The XmlIacpResponse Class

The XmlIacpResponse object extends IacpResponse and adds the getXmlContent() method.

Signature

package **XmlIacpResponse**;

@ISA = ('IacpResponse');

Constructor

```
$myXmlResponse = new XmlIacpResponse($theIacpResponse);
```

Will throw a DOCIntranet::BadContentTypeException if the content-type of the response is specified and is not "text/xml".

Parameters:

response - The IacpResponse obtained from the IacpClient->getResponse() method.

Methods

setResponse

```
$myXmlResponse->setResponse($myIacpResponse);
```

Throws DOCIntranet::BadContentTypeException if the content-type of the response is specified and is not "text/xml". Note that the contents are set before the Exception is thrown, so it is still possible to call the getXmlContent() method, which you would expect to throw a parsing Exception. This is included as a convenience in case the IACP service is improperly implemented.

getXmlContent

```
$myXmlDocument = $myXmlResponse->getXmlContent();
```

Returns the XML response. This will be an object to be determined by implementation.

Throws DOCIntranet::XMLParseException if it is unable to parse the delivered XML.

This is just a placeholder in the current XmlIacpResponse specification. The actual exceptions thrown will be implementation dependent.

6.4 The IacpService Class

The IacpService object describes an IACP service provided by an application. An array of IacpService objects is returned by the IacpClient->getCatalog() method. Each object is the application's description of a single IACP service it offers.

Signature

package **IacpService**;

@ISA = ('UNIVERSAL');

Constructor

IacpService

```
$myIacpService = new IacpService(
   [$name,
   description[,
   returnType,
   returnDtd,
   parameters]]);
```

Methods

getName

```
$name = $myIacpService->getName();
```

Returns the name of the service.

getDescription

```
$description = $myIacpService->getDescription();
```

Returns a description of the service.

getReturnType

```
$returnType = $myIacpService->getReturnType();
```

Returns the content-type of the service. Note that content-type is optional to the catalog entry. If it is not specified, this method returns null.

getReturnDtd

```
$returnDtd = $myIacpService->getReturnDtd();
```

Returns:

The ReturnDTD of the service. Note that this is the string returned by the catalog: it has not been checked for existence or validity. Note also that returnDTD is optional to the catalog. If it is not specified, this method returns null.

getParameters

```
%passedParams = $myIacpService->getParameters();
```

Returns:

The parameters accepted by the service as input. Note that parameters are optional to the service. If the service does not accept parameters, this method returns null.

6.5 The BadContentTypeException Class

Signature

package **BadContentTypeException**;

@ISA = ('Exception');

Constructor

```
$exc = new BadContentTypeException();
```

Constructs a BadContentTypeException.

6.6 The XmlParseException Class

Signature

package **XmlParseException**;

@ISA = ('Exception');

Constructor

```
$exc = new XmlParseException();
```

Constructs a XmlParseException.

6.7 The IacpConnectionException Class

Signature

package **IacpConnectionException**;

@ISA = ('Exception');

Constructor

```
$exc = new IacpConnectionException();
```

Constructs a IacpConnectionException.

6.8 The IacpAuthenticationException Class

Signature

package **IacpConnectionException**;

@ISA = ('Exception');

Constructor

```
$exc = new IacpConnectionException();
```

Constructs an IacpConnectionException.

7 The IACP Server API for Perl

The IACP server API for Perl is also in DOCIntranet.pm. It uses the CPAN module Exception.

The primary class is **IacpHandler**. You can use it to get the parameters and relevant headers of the request, set the authentication scheme and check for authorization, and to return a variety of failure responses.

7.1 The IacpHandler Class

An IacpHandler object is used to process requests from other applications via IACP. The following depicts a typical use of an IacpHandler.

```
$myHandler = new IacpHandler();


try {
    $myHandler->examineRequest();
}


catch DOCIntranet::SomeException exc =>
 sub {
      # handle it
      return;
 }


# This app may want to restrict access or customize response based
# on the identity of the requesting app.


$reqIid = $myHandler->getRequesterIid();


# We can request paramaters individually by name or all at once.
```

```
%params = $myHandler->getParameters();

$reqService = $myHandler->getReqestedService();
if ($reqService cmp 'catalog') {

    # We have decided not to require authentication for the catalog
    # service.

    $reqService->setAuthenticationScheme($AUTH_NONE);

    # Note that myCatalogGenerationMethod sets the response content-type
    # to "text/xml".

    &myCatalogGenerationMethod(...);
    return;
}

# The rest of our services use $AUTH_IACP_TIME_STAMP, so we can check
# authentication here rather than individually per service.

$myHandler->setAuthenticationScheme($AUTH_IACP_TIME_STAMP);
if (!$myHandler->isAuthorized()) {

    # HTTP response code 401

    $myHandler->demandAuthentication();
    return;
```

```
        }

    if ($reqService cmp 'StringService1') {

         # The catalog service has provided a ReturnType for this service of
         # "text/plain".  The myStringService1ResponseMethod should set the
         # response content-type to "text/plain".

         &myStringService1ResponseMethod(...);
         return;

    } elsif ($reqService cmp 'XmlService2') {

         # The catalog service has provided a ReturnType for this service of
         # "text/xml".  The myXmlService2ResponseMethod should set the
         # response content-type to "text/xml".

         &myXmlServiceResponse2Method(...);
         return;
    } else {

        # HTTP response code 404

        $myHandler->returnNotFoundError();
    }
```

Signature

package **IacpHandler**;

@ISA = ('UNIVERSAL');

Variables

$AUTH_NONE

Authentication scheme: Instructs the IacpHandler to ignore authentication.

$AUTH_BASIC

public static final int AUTH_BASIC

Authentication scheme: Instructs the IacpHandler to use Basic authentication as described in [RFC 2617].

$AUTH_DIGEST

Authentication scheme: Instructs the IacpHandler to use Digest authentication as described in [RFC 2617].

$AUTH_IACP_SIMPLE

Authentication scheme: Instructs the IacpHandler to use the IACP authentication as described in the protocol description. Neither time stamp or single-use techniques are to be employed.

$AUTH_IACP_TIME_STAMP

Authentication scheme: Instructs the IacpHandler to include a timestamp in the authentication nonce and check that each returned nonce is no older than the age set by setMaxNonceAge().

$AUTH_IACP_SINGLE_USE

Authentication scheme: Instructs the IacpHandler to keep track of nonces so that it can confirm that each nonce is used only once. This is not likely to be implemented in the IACP version 1.0 library, and may throw an IllegalArgumentException.

$AUTH_IACP_SINGLE_USE_TIME_STAMP

Authentication scheme: Combines AUTH_IACP_SINGLE_USE and AUTH_IACP_TIME_STAMP. This is not likely to be implemented in the IACP version 1.0 library, and may throw an IllegalArgumentException.

Constructor

`$myIacpHandler = new IacpHandler($docDirectory);`

Parameters:

docDirectory - A DocDirectory object.

Methods

---

examineRequest

`$myIacpHandler->examineRequest();`

Causes the IacpHandler to determine that sufficient information about the request is available to respond to subsequent method calls.

Throws exceptions to be determined at implementation.

---

setMaxNonceAge

`$myIacpHandler->setMaxNonceAge($age);`

Specifies the maximum age of a returned authentication nonce. Meaningful only if a time-stamp authentication scheme is used.

Parameters:

age - The maximum acceptable age of the nonce in seconds. If the parameter is not set, it defaults to 60.

---

setAuthenticationScheme

`$myIacpHandler->setAuthenticationScheme($scheme);`

Specifies the authentication scheme to be enforced by the handler. Default is $AUTH_IACP_TIME_STAMP.

isAuthorized

```
$auth = $myIacpHandler->isAuthorized();
```

Indicates that the request meets the requirements of the specified authentication scheme. At the time that it is called, it checks that the provided Authorization passes the currently specified authentication scheme. For IACP authorization schemes, this includes performing asymmetric decryption.

If the current authentication scheme is $AUTH_NONE, the method returns true.

If false is returned, the IacpHandler must provide a response to the requester containing HTTP response code 401 (Unauthorized) and a WWW-Authenticate header specifying the Authorization required of the request. To do so, call IacpHandler's demandAuthentication method.

If a service requires a greater level of authorization than IACP authentication provides, it can use the requester's IID to decide if or what it should provide in the response. Should the request fail the application's requirements even though it provided IACP authentication, call the IacpHandler's returnNoAccessError method.

getRequesterIid

```
$reqIid = $myIacpHandler->getRequesterIid();
```

Returns the IID of the requesting application.

getRequestedService

```
$serviceName = $myIacpHandler->getRequestedService();
```

Returns the name of the requested service. For instance, if the catalog service is requested, 'catalog'.

getParameter

```
$paramValue = $myIacpHandler->getParameter($paramName);
```

Returns the value supplied for the paramName parameter. If the parameter was not included in the request, returns null.

Parameters:

paramName - The parameter for which a value is sought.

---

getParameters

`%params = $myIacpHandler->getParameters();`

Returns:

A hash containing the parameters supplied by the requester. If no parameters were supplied by the requester, returns null.

---

demandAuthentication

`$myHandler->demandAuthentication();`

Respond to the requester that they are required to authenticate to access the requested service. Causes the IacpHandler to respond with HTTP response code 401 (Unauthorized) and the appropriate WWW-Authenticate header given the specified authentication scheme.

---

returnNoAccessError

`$myHandler->returnNoAccessError();`

Respond to requester that they have successfully authenticated, but are not allowed access to the requested service. IacpHandler responds with HTTP response code 403 (Forbidden) .

---

returnNotFoundError

`$myHandler->returnNotFoundError();`

Respond to requester that the requested service is not available. IacpHandler responds with HTTP response code 404 (Not Found).

7.2 The IllegalArgumentException Class

Signature

package **IllegalArgumentException**;

@ISA = ('Exception');

Constructor

`$exc = new IllegalArgumentException();`

Constructs an IllegalArgumentException.

7.3 The NotInitializedException Class

Signature

package **NotInitializedException**;

@ISA = ('Exception');

Constructor

`$exc = new NotInitializedException();`

Constructs an NotInitializedException.

---

8 The IACP Client API for PHP

The IACP client API for PHP is in library file DOCIntranet.inc. The primary class is **IacpClient**. You can discover an application's services via its `getCatalog` method. It returns an array of **IacpService** objects. From these you can obtain the names and details of the application's services. If you want to request from one of these services, the **IacpClient** provides `getResponse()` methods. The **IacpClient** handles whatever authentication is demanded by the service transparently.

8.1 The IacpClient Class

An IacpClient object is used to request information from other applications via IACP. The following depicts a typical use of an IacpClient.

```
include "DOCIntranet.inc";


$myClient = new IacpClient($MY_IID, $MY_PRIVATE_KEY);

$myClient->setServiceUrl($theServiceUrl);

/***********************************************************/

/* For a binary contents you can use IacpResponse directly. */

/***********************************************************/


// (set appropriate service parameters in $paramArray.)

$myResponse =

    $myClient->getResponse("aBinaryService", $paramArray);


isSet($myResponse) OR switch($doc_errno)

{

    /* We would test for case 0 if the function could intentionally

    ** return a null.  This function will never intentionally
    ** return a null, so we skip it.
    */
    case CONNECTION_ERROR:
        // handle it, possibly return from this function
        break;
    case AUTHENTICATION_ERROR:
        // handle it, possibly return from this function
        break;
}

/* Now get the bytes for use. */

$myBytes = $myResponse->getBytes();

/*************************************************************/
/* For other return types, use a subclass of IacpResponse. */
/*************************************************************/

/* Get an XmlIacpResponse. */

// (set appropriate service parameters in $paramArray.)

$myResponse =
    $myClient->getResponse("anXmlService", $paramArray);

isSet($myResponse) OR switch($doc_errno)
{
    /* We would test for case 0 if the function could intentionally
    ** return a null.  This function will never intentionally
    ** return a null, so we skip it.
    */
    case CONNECTION_ERROR:
        // handle it, possibly return from this function
        break;
    case AUTHENTICATION_ERROR:
        // handle it, possibly return from this function
        break;
}

$myXmlResponse = new XmlIacpResponse($myResponse);
isSet($myXmlResponse) OR switch($doc_errno)
{
    /* We would test for case 0 if the function could intentionally
    ** return a null.  This function will never intentionally
    ** return a null, so we skip it.
    */
    case XML_ERROR:
        // handle it, possibly return from this function
        break;
}

$myXmlObject = $myXmlResponse->getXmlObject();
```

Signature

class **IacpClient**;

Constructor

`$myIacpClient = new IacpClient(string myIid, string myPrivateKey);`

Parameters:

myIid - My application IID.

myPrivateKey - The private key for my application. The IacpClient uses the private key to use a service that requires authentication. Here it is declared a string, but in implementation it may be an object.

Methods

setServiceUrl

`void $myIacpClient->setServiceUrl(string theServiceUrl);`

Parameters:

serviceUrl - The URL of the IACP service from whom information will be requested. Here it is declared a string, but in implementation it may be an object.

---

getCatalog

`IacpService $servicesArray = $myIacpClient->getCatalog();`

Returns:

An array of IacpService objects. Each IacpService object describes a service available from an application providing IACP services.

---

getResponse

```
IacpResponse $myIacpResponse =

    $myIacpClient->getResponse(string serviceName, array params);
```

Parameters:

serviceName - The name of the IACP service to be requested.

params - The parameters to be used as input to the IACP service. The keys are the param names.

Returns:

An IacpResponse object. Unless the IacpResponse content is binary, it will probably be used to initialize one of its subclasses, such as XmlIacpResponse.

8.2 The IacpResponse Class

The IacpResponse object is used to obtain the information in the reply from an IACP service request. It is returned by the IacpClient.getResponse method. If the information in the reply is binary or a plain string, the IacpResponse->getContent() method can be used to obtain the information in binary form or as a string.

Subclasses of IacpResponse exist to facilitate obtaining a response of another content-type. If your application wishes to provide a response in some proprietary or encoded form, you can create a subclass of IacpResponse to help the consumer of of your IACP service information.

The following demonstrates the use of IacpResult for obtaining binary data.

```
include "DOCIntranet.inc";


// (set appropriate service parameters in $paramArray.)


$myResponse =

    $myClient->getResponse("aBinaryService", $paramArray);


isSet($myResponse) OR switch($doc_errno)

{

    /* We would test for case 0 if the function could intentionally

    ** return a null.  This function will never intentionally

    ** return a null, so we skip it.

    */

    case CONNECTION_ERROR:
        // handle it, possibly return from this function
        break;
    case AUTHENTICATION_ERROR:
        // handle it, possibly return from this function
        break;
}

/* Now get the bytes for use. */

$myBytes = $myResponse->getBytes();
```

The following demonstrates use of the XmlIacpResult subclass of IacpResult.

```
    include "DOCIntranet.inc";

    // (set appropriate service parameters in $paramArray.)

    $myResponse =
        $myClient->getResponse("anXmlService", $paramArray);

    isSet($myResponse) OR switch($doc_errno)
    {
        /* We would test for case 0 if the function could intentionally
        ** return a null.  This function will never intentionally
        ** return a null, so we skip it.
        */
        case CONNECTION_ERROR:
            // handle it, possibly return from this function
            break;
        case AUTHENTICATION_ERROR:
            // handle it, possibly return from this function
            break;
    }
```

Signature

class **IacpResponse**;

Constructor

`$myResponse = new Response(string contentType, string content);`

Parameters:

contentType - The value of the content-type header field of the IACP service response.

content - The content of the IACP service response.

Methods

getContentType

`string $theContentType = $myResponse->getContentType();`

Returns the value of the content-type header field of the IACP service response.

getContent

`string $theContent = $myResponse->getContent();`

Returns the contents of the IACP service response.

8.3 The XmlIacpResponse Class

The XmlIacpResponse object extends IacpResponse and adds the getXmlContent() method.

Signature

class **XmlIacpResponse** extends IacpResponse;

Constructor

`$myXmlResponse = new XmlIacpResponse(IacpResponse theIacpResponse);`

Will set $doc_errno to BAD_CONTENT_TYPE if the content-type of the response is specified and is not "text/xml".

Parameters:

response - The IacpResponse obtained from the IacpClient->getResponse() method.

Methods

getXmlContent

`object $myXmlDocument = $myXmlResponse->getXmlContent();`

Returns the XML response. This will be an object to be determined by implementation.

Will set $doc_errno to one of possibly several error codes to be determined at implementation if there is a problem processing the returned XML.

8.4 The IacpService Class

The IacpService object describes an IACP service provided by an application. An array of IacpService objects is returned by the IacpClient->getCatalog() method. Each object is the application's description of a single IACP service it offers.

Signature

package **IacpService**;

Constructor

IacpService

```
$myIacpService = new IacpService(
    string name,
   string description,
   string returnType,
   string returnDtd,
   array parameters);
```

Methods

---

getName

```
string $name = $myIacpService->getName();
```

Returns the name of the service.

---

getDescription

```
string $description = $myIacpService->getDescription();
```

Returns a description of the service.

---

getReturnType

```
string $returnType = $myIacpService->getReturnType();
```

Returns the content-type of the service. Note that content-type is optional to the catalog entry. If it is not specified, this method returns null.

---

getReturnDtd

```
string $returnDtd = $myIacpService->getReturnDtd();
```

Returns:

The ReturnDTD of the service. Note that this is the string returned by the catalog: it has not been checked for existence or validity. Note also that returnDTD is optional to the catalog. If it is not specified, this method returns null.

getParameters

```
array $paramArray = $myIacpService->getParameters();
```

Returns:

The parameters accepted by the service as input. Note that parameters are optional to the service. If the service does not accept parameters, this method returns null.

9 The IACP Server API for PHP

The IACP server API for PHP is also in DOCIntranet.inc.

The primary class is **IacpHandler**. You can use it to get the parameters and relevant headers of the request, set the authentication scheme and check for authorization, and to return a variety of failure responses.

9.1 The IacpHandler Class

An IacpHandler object is used to process requests from other applications via IACP. The following depicts a typical use of an IacpHandler.

```
$myHandler = new IacpHandler();

$myHandler->examineRequest();

switch($doc_errno)

{

    case SOME_ERROR:

        // handle it, possibly return from this function

        break;

}


/*

** This app may want to restrict access or customize response based

** on the identity of the requesting app.

*/

$reqIid = $myHandler->getRequesterIid();

/* We can request paramaters individually by name or all at once. */


$params = $myHandler->getParameters();
```

```
$reqService = $myHandler->getReqestedService();
if ($reqService == "catalog")
{
    /*
    ** We have decided not to require authentication for the catalog
    ** service.
    */
    $reqService->setAuthenticationScheme(AUTH_NONE);

    /*
    ** Note that myCatalogGenerationMethod sets the response content-type
    ** to "text/xml".
    */
    myCatalogGenerationMethod(...);
    return;
}

/*
** The rest of our services use AUTH_IACP_TIME_STAMP, so we can check
** authentication here rather than individually per service.
*/
$myHandler->setAuthenticationScheme(AUTH_IACP_TIME_STAMP);
if (!$myHandler->isAuthorized())
{
    // HTTP response code 401

    $myHandler->demandAuthentication(); // does not return
    return;
```

```
}

if ($reqService == "StringService1")
{
    /*
    ** The catalog service has provided a ReturnType for this service of
    ** "text/plain".  The myStringService1ResponseMethod should set the
    ** response content-type to "text/plain".
    */
    myStringService1ResponseMethod(...);
    return;
}

elseif ($reqService == "XmlService2")
{
    /*
    ** The catalog service has provided a ReturnType for this service of
    ** "text/xml".  The myXmlService2ResponseMethod should set the
    ** response content-type to "text/xml".
    */
    myXmlServiceResponse2Method(...);
    return;
}

else
{
    # HTTP response code 404
    $myHandler->returnNotFoundError(); // does not return
    }
```

Signature

class **IacpHandler**;

Constants

AUTH_NONE

Authentication scheme: Instructs the IacpHandler to ignore authentication.

AUTH_BASIC

public static final int AUTH_BASIC

Authentication scheme: Instructs the IacpHandler to use Basic authentication as described in [RFC 2617].

AUTH_DIGEST

Authentication scheme: Instructs the IacpHandler to use Digest authentication as described in [RFC 2617].

AUTH_IACP_SIMPLE

Authentication scheme: Instructs the IacpHandler to use the IACP authentication as described in the protocol description. Neither time stamp or single-use techniques are to be employed.

AUTH_IACP_TIME_STAMP

Authentication scheme: Instructs the IacpHandler to include a timestamp in the authentication nonce and check that each returned nonce is no older than the age set by setMaxNonceAge().

AUTH_IACP_SINGLE_USE

Authentication scheme: Instructs the IacpHandler to keep track of nonces so that it can confirm that each nonce is used only once. This is not likely to be implemented in the IACP version 1.0 library, and may throw an IllegalArgumentException.

AUTH_IACP_SINGLE_USE_TIME_STAMP

Authentication scheme: Combines AUTH_IACP_SINGLE_USE and AUTH_IACP_TIME_STAMP. This is not likely to be implemented in the IACP version 1.0 library, and may throw an IllegalArgumentException.

Constructor

`$myIacpHandler = new IacpHandler($docDirectory);`

Parameters:

docDirectory - A DocDirectory object.

Methods

examineRequest

`void $myIacpHandler->examineRequest();`

Causes the IacpHandler to determine that sufficient information about the request is available to respond to subsequent method calls.

Sets $doc_errno on errors to be determined at implementation.

setMaxNonceAge

`void $myIacpHandler->setMaxNonceAge(int age);`

Specifies the maximum age of a returned authentication nonce. Meaningful only if use a time-stamp authentication scheme.

Parameters:

age - The maximum acceptable age of the nonce in seconds. If the parameter is not set, it defaults to 60.

setAuthenticationScheme

```
void $myIacpHandler->setAuthenticationScheme(int scheme);
```

Specifies the authentication scheme to be enforced by the handler. Default is AUTH_IACP_TIME_STAMP.

isAuthorized

```
bool $auth = $myIacpHandler->isAuthorized();
```

Indicates that the request meets the requirements of the specified authentication scheme. At the time that it is called, it checks that the provided Authorization passes the currently specified authentication scheme. For IACP authorization schemes, this includes performing asymmetric decryption.

If the current authentication scheme is AUTH_NONE, the method returns true.

If false is returned, the IacpHandler must provide a response to the requester containing HTTP response code 401 (Unauthorized) and a WWW-Authenticate header specifying the Authorization required of the request. To do so, call IacpHandler's demandAuthentication method.

If a service requires a greater level of authorization than IACP authentication provides, it can use the requester's IID to decide if or what it should provide in the response. Should the request fail the application's requirements even though it provided IACP authentication, call the IacpHandler's returnNoAccessError method.

getRequesterIid

```
string $reqIid = $myIacpHandler->getRequesterIid();
```

Returns the IID of the requesting application.

getRequestedService

```
string $serviceName = $myIacpHandler->getRequestedService();
```

Returns the name of the requested service. For instance, if the catalog service is requested, "catalog".

getParameter

`string $paramValue = $myIacpHandler->getParameter(string paramName);`

Returns the value supplied for the paramName parameter. If the parameter was not included in the request, returns null.

Parameters:

paramName - The parameter for which a value is sought.

getParameters

`array $params = $myIacpHandler->getParameters();`

Returns:

An array containing the parameters supplied by the requester. If no parameters were supplied by the requester, returns null. The keys of the array are the parameter names.

demandAuthentication

`void $myHandler->demandAuthentication();`

Respond to the requester that they are required to authenticate to access the requested service. Causes the IacpHandler to respond with HTTP response code 401 (Unauthorized) and the appropriate WWW-Authenticate header given the specified authentication scheme.

Note that this function exits upon returning the response.

returnNoAccessError

`void $myHandler->returnNoAccessError();`

Respond to requester that they have successfully authenticated, but are not allowed access to the requested service. IacpHandler responds with HTTP response code 403 (Forbidden) .

Note that this function exits upon returning the response.

returnNotFoundError

```
void $myHandler->returnNotFoundError();
```

Respond to requester that the requested service is not available. IacpHandler responds with HTTP response code 404 (Not Found).

Note that this function exits upon returning the response.

10 References

[RFC 2617] - HTTP Authentication: Basic and Digest Access Authentication

http://www.faqs.org/rfcs/rfc2617.html